МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«БРЯНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА И.Г. ПЕТРОВСКОГО»

Институт естественнонаучный институт Факультет физико-математический

Кафедра информатики и прикладной математики

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

(БАКАЛАВРСКАЯ РАБОТА)

«Разработка модуля клиентской части системы учёта рабочего времени и графика работ сотрудников организации (на примере работы Информационного центра УМВД России по Брянской области)»

Код, направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Направленность (профиль) программы: Сетевые технологии

|  |  |
| --- | --- |
| *(Рекомендована/Не рекомендована к защите в ГЭК)*  протокол № от « » 20 г. Заведующий кафедрой:  информатики и прикладной математики  Денисов И.А. профессор  (подпись) | Обучающийся Леженников Даниил Олегович  4 курса, 2 группы очная форма обучения    (подпись)  Руководитель Гончаров Кирилл Александрович доктор технических наук, доцент  (подпись) |

Брянск 2025 г

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc196401039)

[ГЛАВА 1. АНАЛИЗ И ЗНАЧИМОСТЬ УЧЕТА РАБОЧЕГО ВРЕМЕНИ В ОРГАНИЗАЦИИ 5](#_Toc196401040)

[1.1. Значение учета рабочего времени для организаций 5](#_Toc196401041)

[1.2 Обзор существующих систем учета рабочего времени 7](#_Toc196401042)

[1.2.1 Системы на основе бумажных носителей 7](#_Toc196401043)

[1.2.2 Автоматизированные системы учета времени 8](#_Toc196401044)

[1.2.3 Мобильные приложения для учета рабочего времени 8](#_Toc196401045)

[1.2.4 Анализ готовых решений для учета рабочего времени 9](#_Toc196401046)

[1.3 Выбор инструментов и технологий 14](#_Toc196401047)

[1.3.1 Выбор языка программирования 14](#_Toc196401048)

[1.3.2 Выбор фреймворка для разработки 15](#_Toc196401049)

[1.3.3 Выбор СУБД для разработки 17](#_Toc196401050)

[ВЫВОДЫ ПО ГЛАВЕ 1 20](#_Toc196401051)

[ГЛАВА 2. РЕАЛИЗАЦИЯ ПРАКТИЧЕСКОЙ ЧАСТИ ПРОЕКТА 21](#_Toc196401052)

[2.1. Структура web-приложения 21](#_Toc196401053)

[2.2. Реализация функций web-приложения 37](#_Toc196401054)

[2.3 Тестирование 43](#_Toc196401055)

[ВЫВОДЫ ПО ГЛАВЕ 2 49](#_Toc196401056)

[ЗАКЛЮЧЕНИЕ 51](#_Toc196401057)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 53](#_Toc196401058)

# **ВВЕДЕНИЕ**

Управление временем и трудозатратами сотрудников стало одной из ключевых задач для организаций всех масштабов в условиях современного бизнеса. Эффективное использование рабочего времени играет важную роль в достижении высоких результатов, и для этого необходимо не только оптимизировать внутренние процессы, но и учитывать множество факторов, влияющих на производительность персонала. В связи с этим становится все более актуальной разработка современных систем учета рабочего времени и графиков работы.

Данная дипломная работа посвящена созданию модуля клиентской части системы учета рабочего времени и расписаний сотрудников с использованием языка программирования Python и фреймворка Django. Основная цель исследования – разработка функционального и интуитивно понятного интерфейса, который обеспечит удобное взаимодействие пользователей с системой.

Цели: Разработка пользовательского модуля системы учета рабочего времени и графика дежурств, обеспечивающего удобное взаимодействие сотрудников с системой для просмотра, анализа и управления их расписанием.

Задачи:

* Проанализировать требования к функционалу клиентской части системы учета рабочего времени и графика дежурств.
* Разработать интерфейс для просмотра графика дежурств, аналитики и экспорта данных.
* Разработать интерфейс для отслеживания рабочего времени и отметки посещений с возможностью экспорта в табель.
* Реализовать механизм уведомлений, информирующих пользователей о предстоящих дежурствах.
* Реализовать в виде чата возможность коммуницирования между пользователем и администратором.
* Обеспечить возможность редактирования личных данных пользователей через личный кабинет.
* Протестировать разработанный модуль на соответствие требованиям удобства, производительности и корректности работы.

Предметом исследования являетсяразработка модуля клиентской части системы учёта рабочего времени и графика работ сотрудников организации.

Объектом исследования является система учета рабочего времени и графика сотрудников организации.

ВКР состоит из введения, двух глав, заключения, списка использованных источников и приложений.

В первой части работы осуществляется анализ существующих подходов к учету рабочего времени, а также рассматриваются особенности проектирования и разработки программного обеспечения с использованием Django.

Во второй части описан реализованный модуль, позволяющий пользователям просматривать, редактировать и добавлять информацию о рабочем времени сотрудников.Разработка модуля клиентской части системы учёта рабочего времени и графика работ сотрудников организации позволит упростить процесс учета рабочего времени, повысить точность данных и улучшить взаимодействие сотрудников с системой. Реализованный интерфейс обеспечит удобный доступ к информации о расписаниях, задачах и уведомлениях, что поможет пользователям эффективно управлять своим временем и выполнять рабочие обязанности. Система также позволит автоматизировать процессы, такие как планирование дежурств и контроль за их выполнением, что приведет к сокращению ошибок и повышению производительности.

# **ГЛАВА 1. АНАЛИЗ И ЗНАЧИМОСТЬ УЧЕТА РАБОЧЕГО ВРЕМЕНИ В ОРГАНИЗАЦИИ**

## **Значение учета рабочего времени для организаций**

Учет рабочего времени представляет собой процесс, анализа времени, затраченного сотрудниками и других данных. Эффективность учета рабочего времени работников влияет на их мотивацию к работе. Система учата рабочего времени позволяет оценивать производительность труда. Возможность учёта рабочего времени способствует оптимизации рабочих процессов: анализ собранных данных позволяет обнаружить менее эффективные участки работы и внедрить нужные меры для их улучшения, что, в свою очередь, положительно сказывается на общей производительности.

Система учёта рабочего времени облегчает грамотное управление трудозатратами, позволяя более рационально планировать рабочие часы сотрудников, минимизировать переработки и сократить вероятность трудовых конфликтов. Одним из важных аспектов управления рабочим временем является составление расписания работы, которое устанавливает начало и окончание рабочего дня для каждого из сотрудников. Правильно организованное расписание оказывает значительное влияние на мотивацию работников и их продуктивность. Гибкость расписания, позволяющая работникам самим выбирать время начала и завершения рабочего дня, способствует повышению удовлетворённости сотрудников и, соответственно, их эффективности. Кроме того, чередование рабочих и выходных дней в соответствии с законодательными нормами помогает предотвращать выгорания и поддерживать как физическое, так и психологическое здоровье работников. [30]

Несмотря на очевидные преимущества таких систем, организации сталкиваются с рядом трудностей. Среди этих трудностей — низкая точность учёта при использовании устаревших методов, вроде бумажных таблиц, а также необходимость автоматизации процессов с привлечением современных технологий. Ещё одной проблемой является отсутствие интеграции систем учёта рабочего времени с другими программами, что влечёт за собой дополнительные затраты времени на обработку данных. Переход на новые инструменты требует обучения сотрудников, что часто становится сложной задачей из-за их привычки работать с устаревшими методами.

Учёт рабочего времени и организация графиков сотрудников являются значимыми элементами успешного управления персоналом. Эти процессы позволяют не только осуществлять контроль за затратами времени каждого работника на выполнение задач, но и оптимизировать трудовые ресурсы компании. Формирование гибких графиков работы отвечает потребностям организации и содействует общей продуктивности. В условиях современного рынка точный учёт рабочего времени становится основой для повышения эффективности работы компании и улучшения взаимодействия между её сотрудниками. [29]

В теоретической части данной работы будет проведено исследование различных подходов к учёту рабочего времени, применяемых в компаниях различных масштабов и особенностей деятельности. Особое внимание будет уделено методам и инструментам, направленным на решение задач по управлению трудозатратами и повышению производительности. Основная цель дипломной работы заключена в разработке практического решения, соответствующего современным требованиям и способствующего совершенствованию процессов учёта рабочего времени и планирования графиков сотрудников.

## **Обзор существующих систем учета рабочего времени**

Технологии и автоматизация продолжают играть важную роль в организации рабочих процессов. Автоматизированные системы увеличивают эффективность, уменьшают количество ошибок и ускоряют выполнение задач, улучшая продуктивность в целом. Современные системы учета делают мониторинг более простым, обеспечивают более точный учет трудозатрат сотрудников и помогают оптимизировать распределение ресурсов внутри компании. [29]

Существуют многочисленные системы для учета рабочего времени, которые находят применение в различных организациях. Инструменты такого рода включают в себя относительно простые скрипты для отслеживания рабочего времени и сложные платформы, интегрированные с разнообразными корпоративными системами. Выбор системы учеты зависит не только от потребностей организации, но и от ее масштабов и структуры. Некоторые системы предлагают функции для учета рабочего времени и составления графиков, тогда как другие сосредоточены на анализе производительности и прогнозировании трудозатрат. [30]

В этой части работы будет проведен детальный анализ различных типов систем учета рабочего времени, а также приведены примеры их успешного внедрения в реальных организациях, что позволит лучше понять их преимущества и особенности использования.

## Системы на основе бумажных носителей

Традиционные системы учета рабочего времени имеют некоторые недостатки. Одним из таких недостатков является ручной учет, когда сотрудники свои рабочие часы на различные бумажные носители. Этот метод учет очень простой, но он делает процесс учета весьма трудоемким и способствует возможному возникновению ошибок. Организации, которые используют такие методы учета сталкиваются с проблемами в точности данных и сложностью их обработки. На такой способ учета уходит очень много времени. По окончанию отчетного периода сотрудникам приходится выполнять ручной анализ, что увеличивает риск ошибок и требует значительные ресурсы.

В организациях, где автоматизация еще не получила признания, работники продолжают использовать такие бумажные табели. Такой подход характерен для маленьких организаций, которые не могут позволить себе системы учета или предпочитают простые системы учета рабочего времени.

## Автоматизированные системы учета времени

Системы автоматизации учета рабочего времени значительно повышают эффективность и точность процесса. Одним из ключевых элементов таких решений являются электронные таблицы и базы данных, которые позволяют хранить информацию в структурированном виде. Например, инструменты вроде Google Sheets обеспечивают не только удобное хранение данных, но и возможность совместного редактирования, что упрощает логирование рабочего времени и создание отчетов. Такие программные продукты делают анализ информации более быстрым и прозрачным, исключая риск ошибок, характерных для ручного учета.

Другим методом автоматизации является использование электронных карт и биометрических систем. Работники могут фиксировать время через магнитные карты или биометрические датчики, такие как сканеры отпечатков пальцев. Этот подход минимизирует возможность манипуляций с данными, обеспечивая объективность и надежность учета. Биометрические системы, в частности, исключают подделку рабочих часов, что особенно важно для компаний, где точность данных критически важна.

## Мобильные приложения для учета рабочего времени

Рост мобильных технологий привел к созданию специализированных приложений для учета рабочего времени, которые поддерживают удаленный формат. Эти решения обеспечивают доступность и гибкость, позволяя сотрудникам фиксировать время через смартфоны, что упрощает управление рабочими часами. Например, инструменты вроде TogglTrack интегрируют геолокацию через GPS, что помогает контролировать местоположение удаленных команд.

Приложения такого типа отслеживают время выполнения задач, и анализируют активность в реальном времени. Например, TogglTrack позволяет мониторить прогресс проектов, что облегчает корректировку рабочих процессов. Геолокация исключает манипуляции с данными, предоставляя объективную информацию о местонахождении сотрудников.

## Анализ готовых решений для учета рабочего времени

When I Work – это решение для учета рабочего времени и планирования рабочих смен, которое помогает компаниям эффективно управлять расписаниями и отслеживать рабочие часы сотрудников.

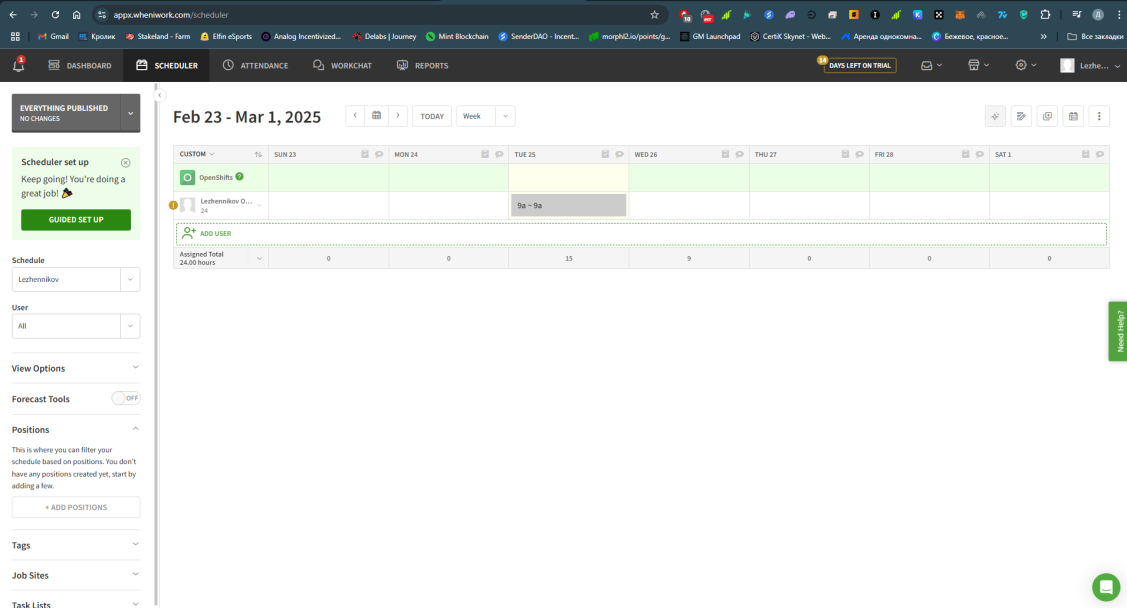


Рисунок 1 – Пользовательский интерфейс сервиса When I Work

When I Work имеет несколько преимуществ. Это удобное планирование смен, которое позволяет сотрудникам легко просматривать и обновлять свои смены, что упрощает процесс планирования и снижает вероятность ошибок. Также система мобильна, так как приложения для iOS и Android дают сотрудникам и менеджерам возможность следить за графиками и рабочим временем, а также запрашивать смены. Для сотрудников эта система гибка, так как они могут запрашивать смены или изменения в расписании. When I Work интегрируется с другими инструментами учета рабочего времени, платежными системами и бухгалтерским ПО. В реальном времени можно вносить изменения в графики, что позволяет оперативно реагировать на изменения.

Однако есть и недостатки. Возможности отчетности ограничены по сравнению с более специализированными решениями. Некоторые пользователи могут столкнуться с недостаточной кастомизацией отчетов. Также система не предоставляет глубокой аналитики, что может быть проблемой для более детального анализа рабочих часов или расчета заработной платы, для чего потребуются дополнительные инструменты. Некоторые пользователи сообщают о трудностях при интеграции с более сложными системами учета, такими как бухгалтерские программы или корпоративные решения. Бесплатная версия имеет ограниченный функционал, и для доступа ко всем возможностям нужно приобрести платный план, что может быть нецелесообразным для малых команд.

When I Work хорошо подходит для малого и среднего бизнеса, который ищет простое в использовании и доступное решение для планирования рабочих смен и отслеживания рабочего времени, но для более комплексных аналитических задач или интеграций с крупными системами могут потребоваться дополнительные ресурсы.

TimeCamp – это решение для учета рабочего времени, которое предоставляет удобные инструменты для отслеживания времени работы сотрудников, управления проектами и генерации отчетов.

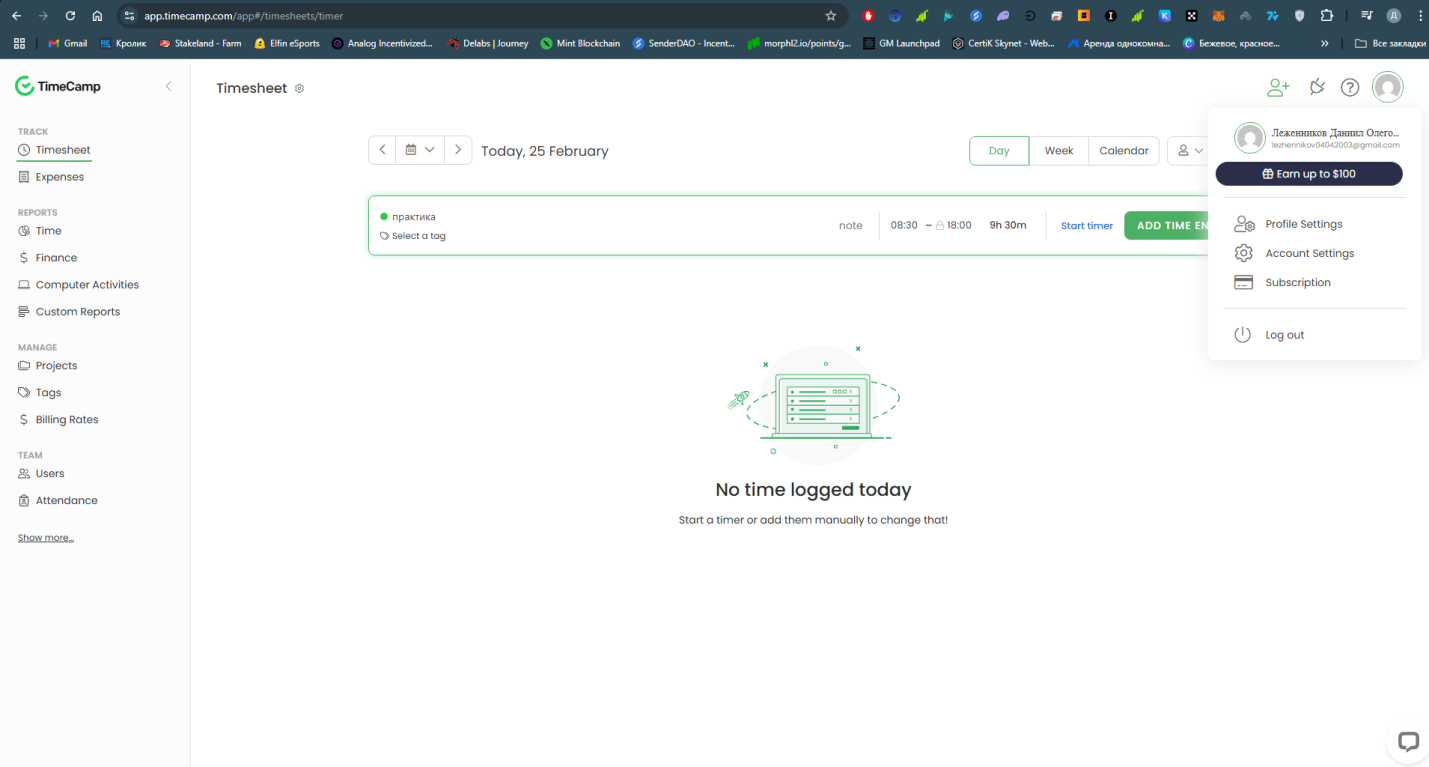


Рисунок 2 – Пользовательский интерфейс сервиса TimeCamp

TimeCamp имеет несколько важных преимуществ. Одним из них является автоматическое отслеживание времени, что упрощает процесс учета рабочего времени и минимизирует ошибки. Система также предлагает гибкие отчеты, которые можно настроить под любые нужды, что полезно для анализа эффективности работы сотрудников. TimeCamp интегрируется с популярными инструментами для управления проектами, такими как Trello, Asana и Jira, что упрощает синхронизацию данных между системами. Она поддерживает работу с несколькими проектами одновременно, что полезно для команд с множеством задач. Также система отлично подходит для отслеживания времени удаленных сотрудников, что важно для компаний с гибкими условиями труда. Простота интерфейса делает использование TimeCamp доступным и понятным для всех сотрудников без необходимости долгого обучения.

Тем не менее, у TimeCamp есть и свои минусы. Бесплатная версия ограничена в функционале, и для доступа ко всем возможностям необходимо приобрести платный план. Также система может быть сложной для очень крупных организаций с множеством сложных задач, поскольку для них TimeCamp может оказаться недостаточно гибким и функциональным. В некоторых случаях пользователи сообщают о проблемах с точностью автоматического отслеживания времени, особенно если задача выполняется вручную или с перерывами.

TimeCamp – это удобное и мощное решение для малого и среднего бизнеса. Но для более крупных организаций с потребностью в глубокой кастомизации и дополнительных функциях отчетности могут потребоваться дополнительные настройки.

**TogglTrack** – это популярное приложение для отслеживания времени, которое используется индивидуальными пользователями и командами для мониторинга рабочих процессов и повышения производительности.

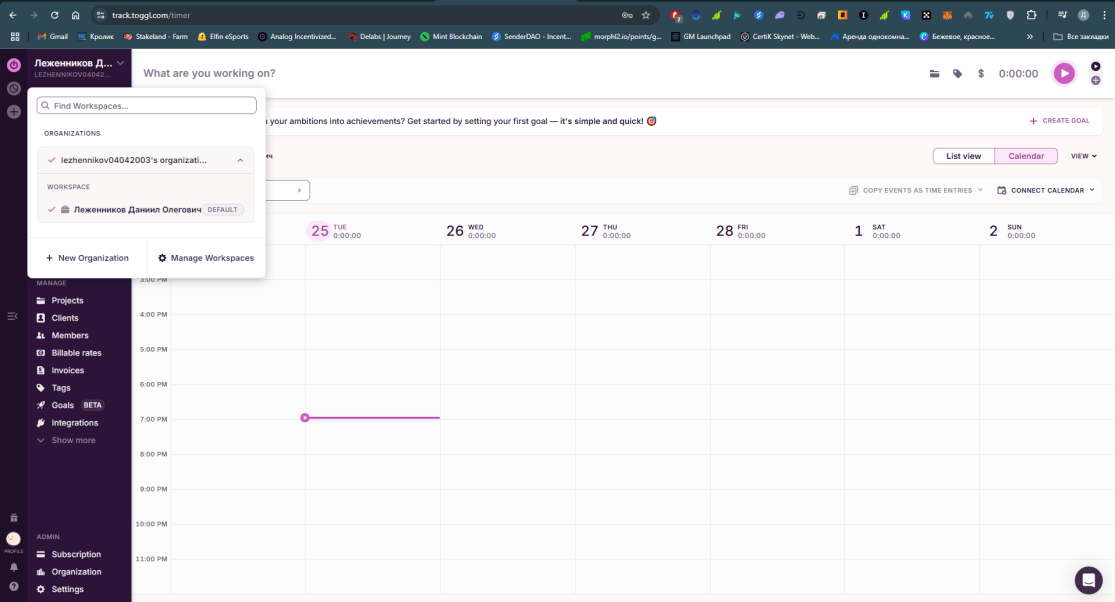


Рисунок 3 – Пользовательский интерфейс сервиса TogglTrack

TogglTrack имеет несколько ключевых преимуществ. Во-первых, система отличается простотой использования: интуитивно понятный интерфейс позволяет легко начать отслеживать время. Также она гибка и подходит как для фрилансеров, так и для команд, работающих над множеством проектов. TogglTrack поддерживает интеграцию с различными инструментами, такими как Asana, Trello и Slack, что облегчает синхронизацию данных между системами. Система также предоставляет подробную аналитику и отчеты о затраченном времени на проекты и задачи, а функция автоматического отслеживания времени делает процесс еще более удобным.

Однако у TogglTrack есть и минусы. Бесплатная версия имеет ограниченный функционал, и для доступа ко всем возможностям требуется платная подписка. Также система может не всегда точно отслеживать время, особенно при работе в фоновом режиме, что может привести к небольшим неточностям. Платная версия может быть дорогой для некоторых пользователей, что делает ее не всегда доступной для всех.

TogglTrack – это удобный инструмент для отслеживания времени, который подходит для малых и средних команд, но для более сложных нужд могут потребоваться дополнительные функции или платные планы.

Проанализировав готовые решения для учета рабочего времени, можно сказать, что все они предлагают удобные и функциональные инструменты для различных типов организаций. В таких системах, как When I Work, TimeCamp и TogglTrack, можно выделить такие положительные моменты, как мобильность, гибкость, интеграции с другими приложениями и возможность получения подробных отчетов. Однако, несмотря на их привлекательные особенности, они имеют значительные ограничения, такие как ограниченный функционал в бесплатных версиях, проблемы с точностью отслеживания времени и высокие расходы на платные планы для более крупных команд.

Пользуясь опытом уже готовых систем учета рабочего времени и графиков, можно создать собственный продукт, максимально адаптированный под нужды компании. Это обеспечит большую гибкость, точность и удобство в управлении трудозатратами, позволяя учитывать уникальные требования бизнеса. Индивидуальная система будет более эффективно решать задачи, такие как специфические отчеты, точное отслеживание времени и интеграция с другими внутренними инструментами. Таким образом, создание собственной системы позволит не только повысить точность учета, но и значительно улучшить управляемость и оперативность процессов.

## **Выбор инструментов и технологий**

Выбор технологий для разработки web-приложения играет важную роль в разработке системы, определяя её возможности, скорость работы и способности к росту. Выбор технологий влияет на то, как система будет справляться с задачами.

## Выбор языка программирования

Веб-разработка предоставляет множество языков программирования, каждый из которых обладает своими особенностями. Выбор подходящего языка определяется задачами проекта и ожиданиями от его эффективности. Например, TypeScript [2], расширяя JavaScript [5], добавляет строгую типизацию, что делает код предсказуемым и надежным. Это важно для крупных приложений, где нужна упорядоченная структура. Однако такая типизация иногда усложняет поддержку кода, особенно при работе со сложной логикой. В аналитических задачах строгая типизация может сильно замедлять процесс разработки и тестирования.

PHP [13] сохраняет свою популярность в разработке благодаря богатому набору фреймворков, таких как Laravel [6] и Symphony [19]. Они упрощают создание стандартных решений, но при работе с большими объемами данных возникают сложности. Для решения данных сложностей часто прибегают к использованию дополнительных библиотек, что усложняет поддержку приложений и снижает их эффективность при высоких нагрузках.

Java в сочетании с фреймворками вроде Spring [16], выделяется отличной масштабируемостью и производительностью, что делает его востребованным в корпоративных системах. Процесс разработки на Java [4] требует длительной настройки и интеграции. Для аналитических функций приходится подключать сторонние библиотеки и модули, что увеличивает время разработки.

Среди этих языков Python [15] был выбран для разработки данного веб-приложения по нескольким ключевым причинам:

* Python обладает простотой и удобством. Его понятный и простой синтаксис ускоряет написание кода. В отличие от Java, где требуется следить за структурой приложения, Python дает гораздо больше свободы и позволяет сосредоточится на решении задач. Для веб-приложений, где требования могут меняться в процессе разработки это особенно важно. [25]
* Python предоставляет разработчикам удобные фреймворки, также как Django, который предоставляет встроенные инструменты для множества задач в веб-разработке, от настройки маршрутов и до работы с базами данных. Это заметно сокращает время на разработку и избавляет разработчика от необходимости писать все с чистого листа.
* Python отлично подходит для работы с данными благодаря мощным библиотекам, таким как Pandas [12], NumPy [11] и Matplotlib [7], которые упрощают процесс анализа больших объемов информации. Возможность экспорта данных в разные форматы делает его незаменимым помощником в решении аналитических задач.
* Python применяется в самых различных сферах. Такая многосторонность помогает создавать приложения, которые легко приспосабливаются к новым условиям и требованиям. Можно внедрять сложные аналитические возможности или связывать новый проект с внешними системами.
* Python благодаря своей простоте и поддержке активного сообщества разработчиков не теряет своей актуальности. Python предлагает множество библиотек и подробную документацию, что способствует ускорению работы и помогает быстрее справляться с трудностями.

Именно эти качества сделали Python выбором для разработки веб-приложения, где нужно обрабатывать данные, проводить анализ и обеспечивать удобное взаимодействие с пользователями.

## Выбор фреймворка для разработки

Выбор фреймворка для разработки веб-приложения требует оценки его возможностей, способности к масштабированию, удобству работы и его надежности. Например, Flask [3] – это микро-фреймворк для Python с минимальным количеством функций. Гибкость является его ключевой особенностью. Это позволяет разработчику самому подбирать необходимые компоненты. Это помогает создавать легкие и быстрые приложения, но для базовых задач, таких как маршрутизация и работа с базами данных, приходится подключать много дополнительных библиотек. Такой подход замедляет разработку и усложняет дальнейшую поддержку, поэтому Flask лучше подходит для небольших проектов или прототипов. Для более сложных задач его может не хватить.

FastAPI [2] — современный фреймворк для Python, созданный специально для разработки API. Он отличается скоростью работы благодаря своей поддержке асинхронности и автоматическому созданию документации. Это делает его отличным выбором для быстрых API приложений, однако для стандартных веб-систем с интерфейсами он менее практичен. Для работы с базами данных необходимо подключать и настраивать внешние библиотеки.

Tornado [20] — асинхронный фреймворк, который хорошо справляется с приложениями, требующими множество одновременных подключений. Данный фреймворк предоставляет низкоуровневые инструменты для обработки запросов, которые обеспечивают высокую производительность. Однако в отличие от Django, Tornado не включает встроенных решений для аутентификации, маршрутизации или взаимодействия с базами данных. Из-за этого добавление и настройка новых функций может замедлить процесс разработки.

При выборе фреймворка для разработки веб-приложения ключевым фактором стала комплексность решений, которые он предлагает. Django был выбран благодаря своему богатому набору встроенных инструментов, что существенно ускоряет процесс создания приложений. В отличие от микро-фреймворков, таких как Flask, он включает готовые компоненты для маршрутизации, работы с базами данных, аутентификации и создания форм. Это особенно важно для задач, требующих структурированного подхода, например, при разработке графика дежурств. [23]

Безопасность приложения стала одним из критериев при выборе фреймворка. Django предоставляет встроенные средства защиты от SQL-инъекций, XSS-атак и CSRF-уязвимостей, что крайне важно для приложения, которые работают с персональными данными.

Модульная структура Django делает его подходящим как для небольших задач, так и для крупных проектов. Разработчик может подстраивать приложение под новые требования, добавляя возможности вроде аналитики или обработки данных.

Дополнительным плюсом является наличие активного сообщества и подробной документации, которые дают доступ к готовым решениям и ускоряют разработку. Django выделяется благодаря сочетанию функциональности, безопасности и удобству.

## Выбор СУБД для разработки

Выбор системы управления базами данных для веб-приложения зависит от скорости работы, способности к расширению и простоте интеграции с другими частями приложения. Важно учитывать объем данных и ожидаемую нагрузку.

PostgreSQL [14] – объектно-реляционная СУБД с открытым кодом, которая отличается высокой производительностью и поддержкой транзакций. Данная СУБД хорошо справляется с большими объемами данных. Ее использование требует организации репликации и защиты от сбоев, что может быть сложным и требовать усилий. Для небольших проектов может оказаться избыточной.

MySQL [10] – одна из самых популярных СУБД с открытым кодом, которая широко применяется в веб-разработке. Она предлагает эффективные инструменты для работы с большими данными, но для небольших проектов настройка и управление MySQL может быть излишне сложной.

MongoDB [9] – это NoSQL база данных, которая работает с неструктурированными данными. Она идеально подходит для проектов, где данные часто меняются или, необходимо обрабатывать большие объемы информации. Благодаря своей гибкости в хранении данных, MongoDB часто используется в приложениях с высокой нагрузкой. Однако если приложение требует четкой структуры данных или работы с SQL-запросами, MongoDB может быть лишней. Для небольших проектов, где нет необходимости в сложной структуре данных, SQLite станет более простым и удобным выбором.

SQLite [17] была выбрана для разработки данного веб-приложения по нескольким причинам, которые делают её наиболее подходящей СУБД для данного проекта:

* Данная СУБД работает без серверной части. Все данные хранятся в одном файле, что делает её установку и интеграцию значительно проще. Такой подход особенно удобен для проектов, где не требуется сложная структура базы данных или высокая нагрузка на сервер.
* SQLite использует минимальное количество аппаратных ресурсов, что делает её подходящей для небольших и средних веб-приложений с ограниченными вычислительными мощностями. В отличие от более сложных СУБД, таких как PostgreSQL или MySQL, SQLite не требует большого объема оперативной памяти или мощных процессоров. Это помогает значительно сократить затраты на эксплуатацию. [18]
* Эта СУБД показывает высокую производительность при работе с небольшими и средними объемами данных. Для проектов, которые требуют выполнения простых операций с реляционными данными, таких как создание таблиц или выполнение SQL-запросов, она предлагает эффективные и быстрые решения. Это делает SQLite отличным выбором для проектов с ограниченными требованиями по объему данных.
* SQLite требует гораздо меньше усилий для обслуживания по сравнению с более сложными СУБД, такими как PostgreSQL или MySQL. Она не нуждается в постоянных настройках, организации репликации или создании регулярных резервных копий. Это делает процесс поддержки гораздо проще, что особенно важно для небольших проектов с ограниченными ресурсами.

SQLite выбран для данного приложения благодаря ее простоте и удобству при интеграции. Данная СУБД потребляет минимальное количество ресурсов. При этом SQLite обладает всеми нужными функциями для хранения и обработки данных.

Комбинация Python, Django и SQLite создает прочную основу, для системы учета рабочего времени. Эти технологии подходят под требования приложения и гарантирует успешную разработку и внедрение в систему.

## **ВЫВОДЫ ПО ГЛАВЕ 1**

Рассмотрены основные аспекты учета рабочего времени и его значимость для организации эффективного управления трудозатратами сотрудников. Было подробно раскрыто, как учет рабочего времени способствует повышению производительности, оптимизации рабочих процессов и соблюдению законодательных норм. Также освещено влияние графика работы на мотивацию и производительность сотрудников, включая гибкость графиков и режим работы в удаленном формате.

Рассмотрены актуальные задачи и проблемы, которые связаны с использованием разнообразных методов учета. Оценены проблемы точности систем учета и трудности интеграции с корпоративными системами. Приведены различные подходы к учету рабочего времени, включая использование мобильных приложений и автоматизированных систем учета. Оговорены преимущества и ограничения существующих решений, таких как When I Work, TimeCamp и TogglTrack, что позволяет выделить особенности, такие как мобильность и гибкость, а также ограничения по функционалу и сложности при интеграции с другими системами.

Выделены основные задачи для разработки более адаптированных решений учета рабочего времени, которые могут улучшить точность учета, оптимизировать рабочие процессы и повысить общую продуктивность организации.

# **ГЛАВА 2. РЕАЛИЗАЦИЯ ПРАКТИЧЕСКОЙ ЧАСТИ ПРОЕКТА**

## **2.1. Структура web-приложения**

Клиентская часть приложения должна быть интуитивно понятной и удобной для пользователя. Доступ к приложению должен быть защищён авторизацией и аутентификацией.

На странице для входа должны быть поля для ввода имени пользователи и его пароля. При успешной авторизации пользователь переходит на главную страницу, а при вводе неверных данных выводится ошибка. Должна быть реализована форма для регистрации пользователей, которая содержит поля для имени пользователя, ФИО, email, пароля и его подтверждения. После заполнения всех полей данные отправляются на проверку, и пользователь получает всплывающее уведомление о том, что его запрос принят. На главной странице у пользователя должны отображаться уведомления о предстоящих дежурствах.

В лично кабинете пользователя будут отображаться его личные данный, которые он указывал при регистрации. Также нужно реализовать возможность редактирования профиля. В разделе с аналитикой будет отображаться данные о количестве дежурств за текущий месяц, квартал и год с переводом в часы, а также будут отображаться дежурства за текущий месяц с возможностью экспортировать в CSV и XLSX. При импорте файла CSV в календари по типу Google Calendar, в нем должны создаваться события, которые будут напоминать о дежурствах. Также пользователи могут проверять были ли дежурство в конкретную дату. Должна быть реализована страница с таймером рабочего дня пользователя, также на этой странице должна быть возможность отметить посещение и импорта данных в табель посещаемости. На этой же странице необходимо реализовать список задач, который пользователь сможет редактировать и помечать задачи как выполненные.

Необходимо реализовать чат между пользователем и администратором, в котором сообщения будут отображаться в хронологическом порядке с сохранением истории диалогов. Для безопасности необходимо использовать защиту от CSRF-атак с помощью токенов, а все данные должны передаваться через HTTPS.

Также необходимо реализовать систему уведомлений, которая автоматически будет создавать напоминания о предстоящих дежурствах за день до их начала.

Клиентская часть веб-приложения будет разрабатываться с использованием фреймворка Django, что обеспечит гибкость и удобство в создании и поддержке сложных систем. Приложение будет разделено на несколько функциональных блоков, каждый из которых будет отвечать за свою задачу, при этом основное внимание будет уделено простоте взаимодействия пользователя с системой учета дежурств.



Рисунок 4 – Диаграмма последовательности: Вход в систему

Все данные пользователей хранятся в модели CustomUser, которая является расширением стандартной модели пользователя Django. В этой модели будут добавлены дополнительные поля, такие как дата рождения, отчество, звание и должность, для создания профилей пользователей. Также будут созданы модели для учета информации о дежурствах и уведомлениях.

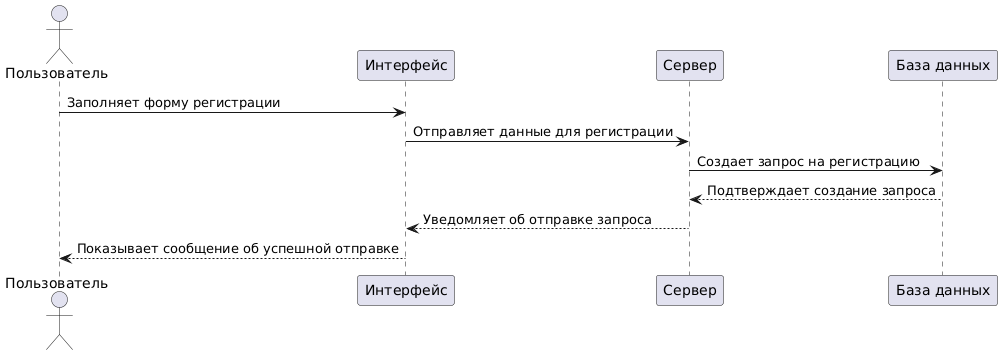


Рисунок 6 – Диаграмма последовательности: Регистрация пользователя

Пользователь сможет просматривать дежурства за весь период, редактировать свои данные и экспортировать график в удобных форматах, таких как CSV и XLSX.

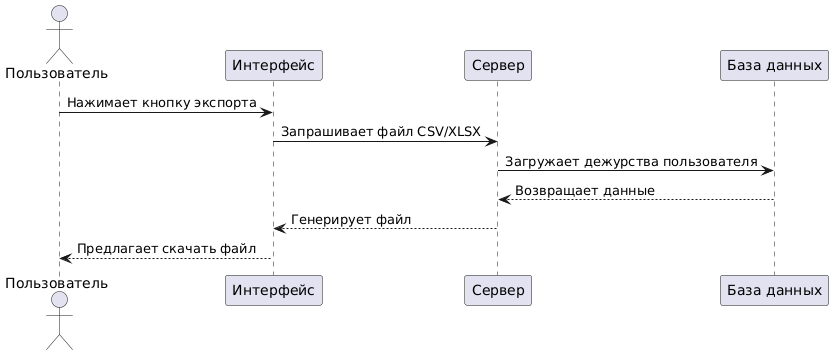


Рисунок 7 – Диаграмма последовательности: Экспорт дежурств в CSV или XLSX

Для анализа активности будет доступна статистика по дежурствам за месяц, квартал и год, а также возможность проверки наличия дежурств на конкретную дату.

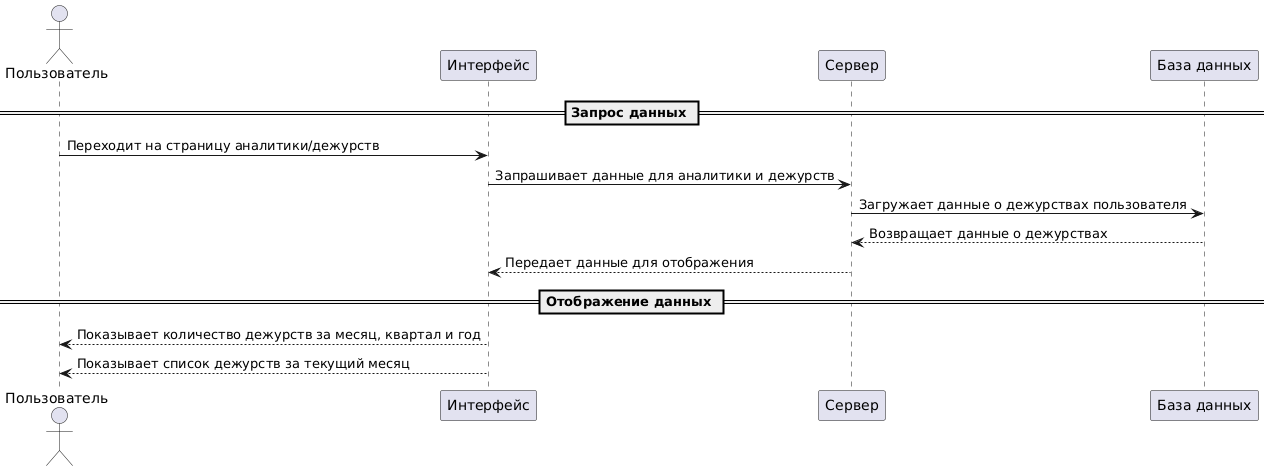


Рисунок 9 –Диаграмма последовательности: Просмотр аналитики по дежурствам

Для качественного управления рабочим временем будет реализован раздел, который включает таймер для отслеживания 8-часового рабочего дня. В дополнение будет доступен список задач, где можно будет добавлять задачи и помечать их как выполненные. Также будет возможность отметить посещение и экспортировать данные в табель учета рабочего времени.

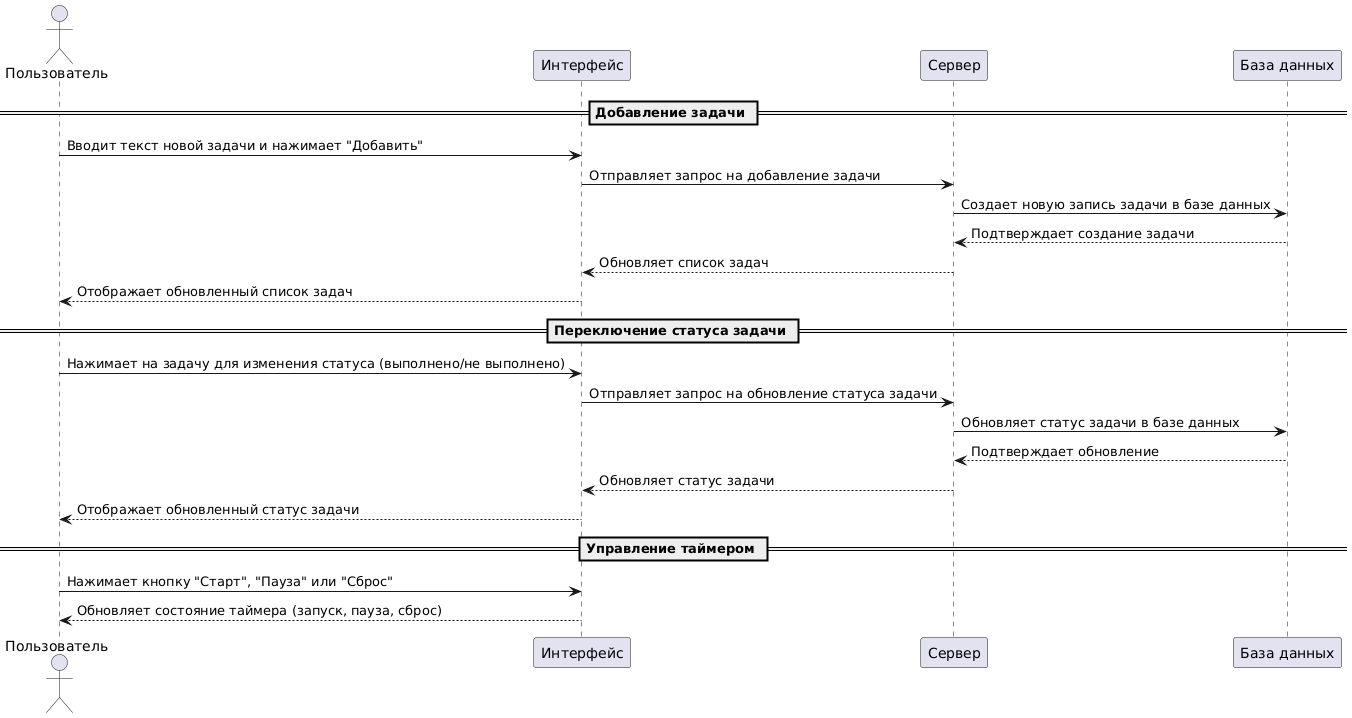


Рисунок 10 – Диаграмма последовательности: Управление рабочим днем и задачами

Для удобного взаимодействия между пользователями и администратором в системе реализован функционал чата. Обычные пользователи могут общаться только с администратором, что обеспечивает четкое разграничение ролей и предотвращает несанкционированные диалоги между пользователями. Чат поддерживает обмен текстовыми сообщениями в реальном времени, при этом все сообщения сохраняются в истории и отображаются в хронологическом порядке.

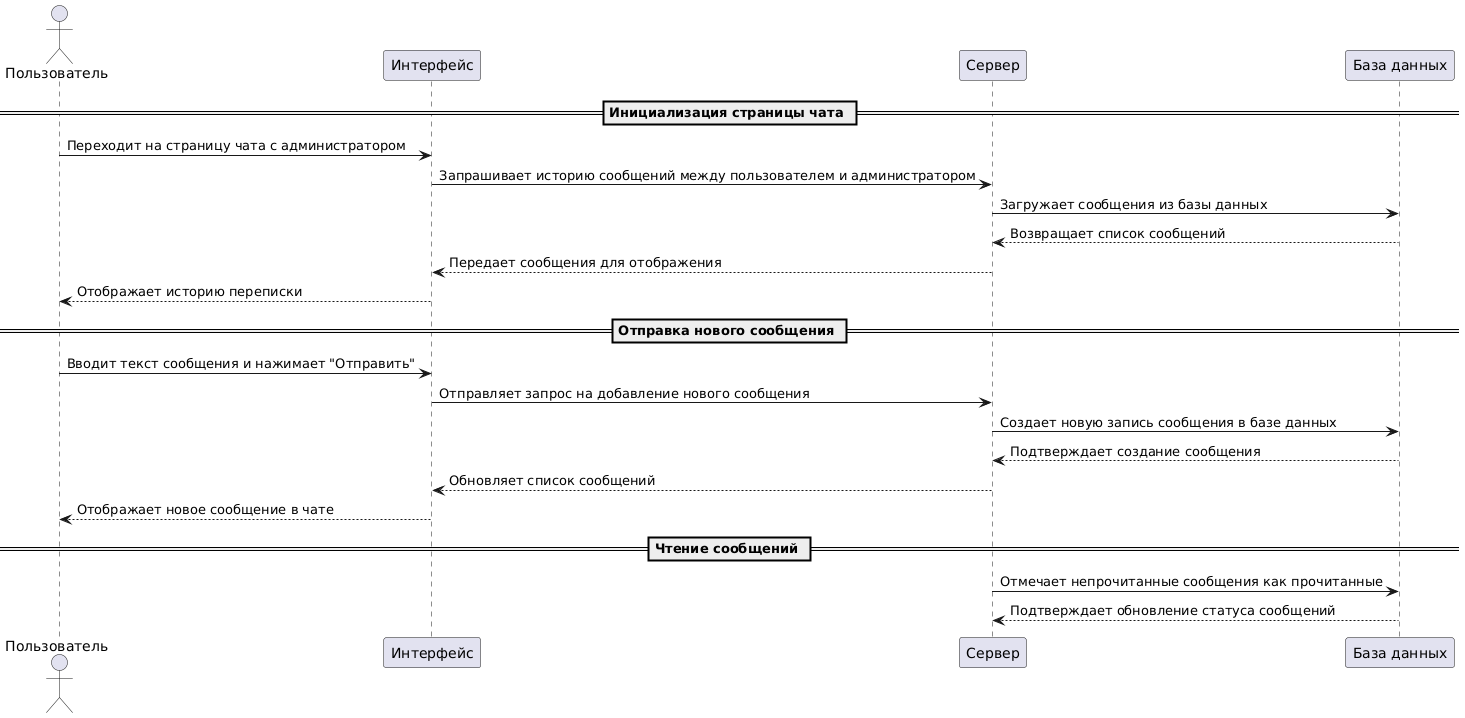


Рисунок 11– Диаграмма последовательности: Чат с администратором

При открытии диалога сообщения будут автоматически помечаться как прочитанные. Данный функционал способствует более эффективному взаимодействию между пользователями и администратором.

Интерфейс приложения будет спроектирован с акцентом на удобство использования, с четкой структурой страниц и логичной организации функциональных блоков. Визуальное оформление будет реализовано с помощью статических файлов CSS и JavaScript, что обеспечит корректное отображение и дополнительные функции, такие как темная тема. Данные будут сохраняться в localStorage, что позволит сохранять информацию при перезагрузке страницы и обеспечит удобную интеграцию с другими модулями приложения.

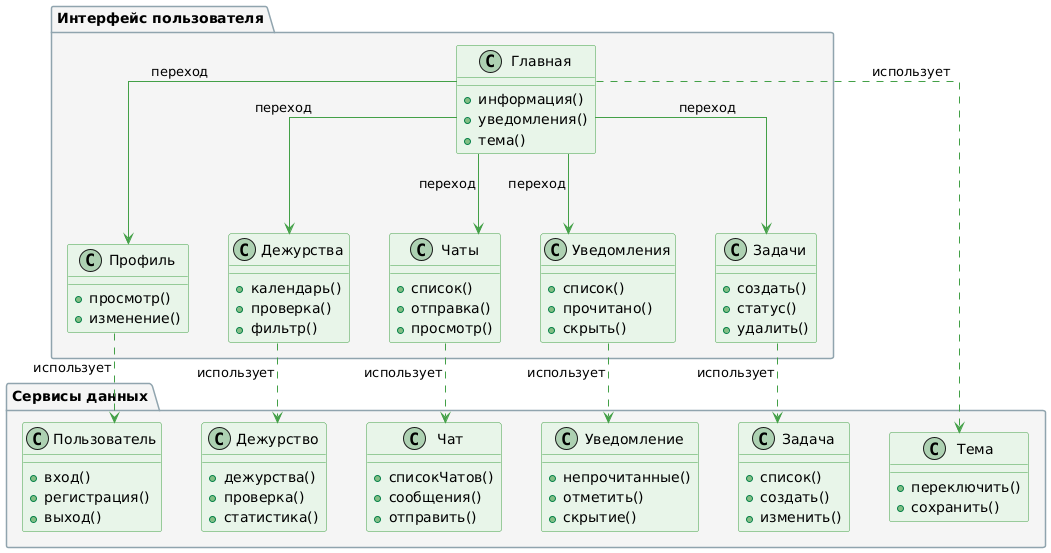


Рисунок 12 – Диаграмма классов

Для реализации всего вышеперечисленного необходимо создать базу данных. Для начала нужно разобрать стандартные таблицы данных. Таблица auth\_group в Django создаётся автоматически для управления правами доступа, она хранит группы пользователей, которые могут быть использованы для назначения ролей, таких как администратор или пользователь. Таблица содержит два поля: id и name (строка длиной до 150 символов, обязательное и уникальное). Эта таблица используется для управления разрешениями в системе аутентификации Django и является частью стандартных таблиц для работы с сессиями и миграциями.

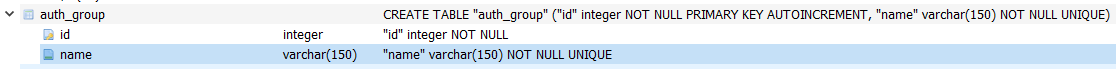


Рисунок 13 – Таблица auth\_group

Таблица django\_content\_type в Django хранит информацию о типах контента (моделях) в приложениях, включая поля id, app\_label и model. Эта таблица используется Django для управления разрешениями и установления связей между моделями, обеспечивая корректную работу с типами контента в проекте.

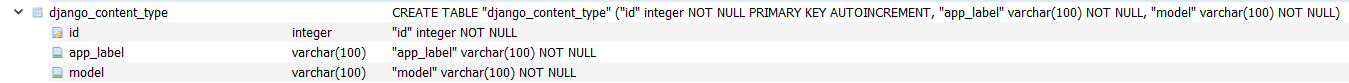


Рисунок 14 – Таблица Django\_content\_type

Таблица django\_session в Django хранит данные сессий пользователей, включая поля session\_key, session\_data и expire\_date. Эта таблица используется для управления сессиями пользователей, поддерживая аутентификацию и состояние, например, для логина и логаута в приложении.

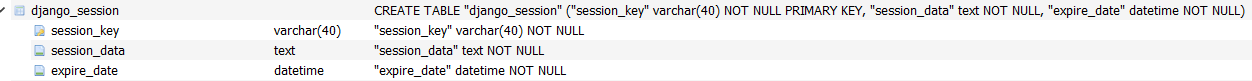


Рисунок 15 – Таблица Django\_session

Теперь нужно рассмотреть пользовательские таблицы, которые основаны на моделях, которые используется в проекте.

Таблица core\_customuser в Django представляет главную модель пользователя, расширяющую AbstractUser и содержащую поля для хранения информации о пользователе, такие как имя, фамилия, email, пароль, дата рождения, должность, звание и другие данные. Она включает в себя несколько связей с другими таблицами, например, через внешние ключи с таблицами core\_changelog, core\_chatmessage, core\_dutyschedule, core\_notification, core\_task, core\_userchangerequest, core\_workattendance и другими. Эти связи позволяют хранить и управлять данными для дежурств, уведомлений, задач и посещаемости. Модель используется для аутентификации в приложении (например, в login\_view и register\_view), отображается в профиле пользователя в user\_dashboard.

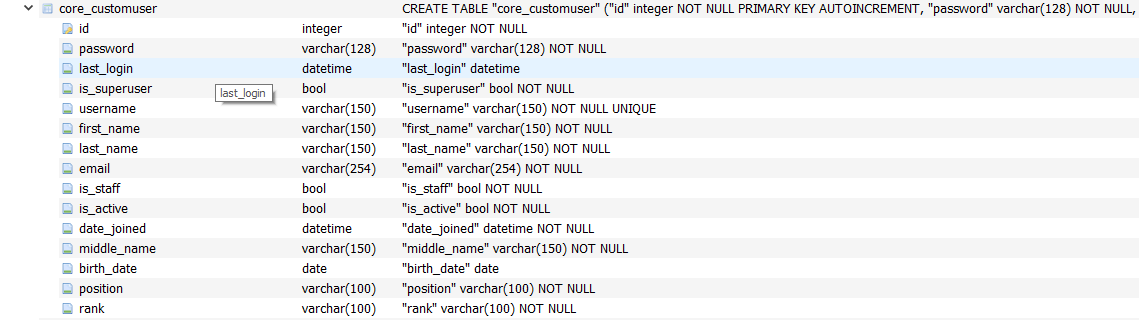


Рисунок 16 – Таблица core\_customuser

Таблица core\_chatmessage в Django хранит сообщения чатов между пользователями и администратором, включая поля: id, message (текст, обязательное поле), timestamp (дата и время отправки сообщения), receiver\_id и sender\_id (ссылки на пользователя через внешний ключ, связывающие отправителя и получателя с таблицей core\_customuser), а также is\_read (логическое поле, указывающее, прочитано ли сообщение). Эта таблица используется для управления чатами между пользователями, например, между обычными пользователями и администраторами, и применяется в представлениях chat\_view, chat\_with\_admin для отображения и обработки сообщений.

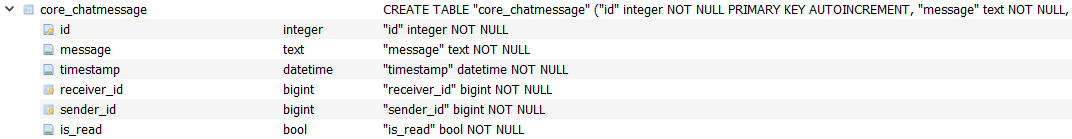


Рисунок 17 – Таблица core\_chatmessage

Таблица core\_dutyschedule в Django хранит расписание дежурств пользователей, включая поля: id, date (дата дежурства, обязательное поле), и user\_id (ссылка на пользователя через внешний ключ, связанная с таблицей core\_customuser). Эта таблица используется для отображения и управления дежурствами в представлениях user\_dashboard, analytics\_view и user\_analytics\_view.

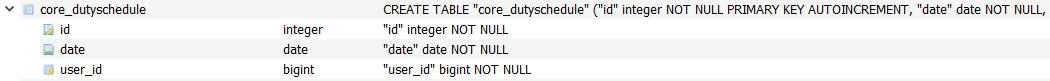


Рисунок 18 – Таблица core\_dutyschedule

Таблица core\_notification в Django хранит уведомления для пользователей, включая поля: id, message (текст уведомления, обязательное поле), is\_read, created\_at (дата и время создания уведомления, обязательное поле) и user\_id (ссылка на пользователя через внешний ключ, связанная с таблицей core\_customuser). Эта таблица используется для создания и управления уведомлениями, такими как напоминания о дежурствах, в представлениях home\_view и mark\_notification\_as\_read.

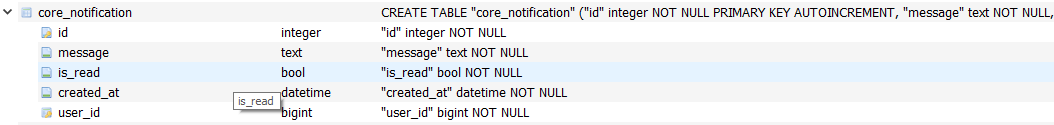


Рисунок 19 – Таблица core\_notification

Таблица core\_task в Django хранит задачи пользователя, включая поля: id, text (текст задачи, обязательное поле), completed (логическое поле, указывающее, выполнена ли задача), date (дата выполнения задачи, обязательное поле) и user\_id (ссылка на пользователя через внешний ключ, связанная с таблицей core\_customuser). Эта таблица используется для управления задачами пользователя на странице "Рабочий день", например, в представлениях workday\_view и функции toggle\_task.

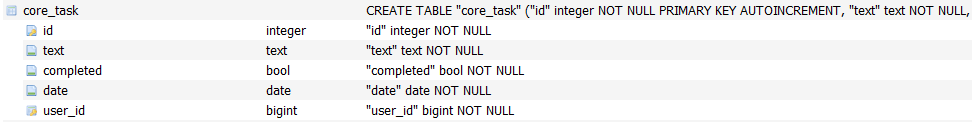


Рисунок 20 – Таблица core\_task

Таблица core\_workattendance в Django хранит данные о посещаемости пользователей, включая поля: id, date (дата посещения, обязательное поле), hours\_worked (количество отработанных часов, обязательное поле), is\_present (логическое поле, указывающее на присутствие пользователя) и user\_id (ссылка на пользователя через внешний ключ, связанная с таблицей `core\_customuser`). Эта таблица используется для отметки посещаемости в представлении `workday\_view` и экспорта данных для табеля посещения.

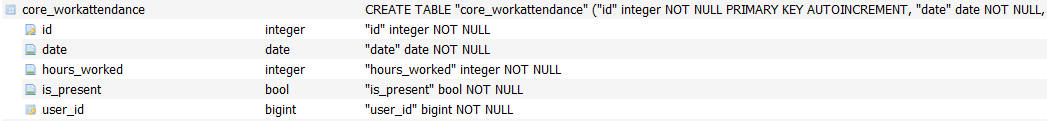


Рисунок 21 – Таблица core\_workattendance

В базе данных связи между таблицами реализуются через внешние ключи (ForeignKey в Django). Центральной таблицей является core\_customuser, которая связана с большинством других таблиц, включая core\_chatmessage, core\_dutyschedule, core\_notification, core\_task, core\_userchangerequest, и core\_workattendance, через поля, такие как user\_id или аналогичные.

Пользовательские таблицы составляют основу функционала проекта. Они позволяют эффективно управлять данными о пользователях, их дежурствах, задачах, уведомлениях и посещаемости, обеспечивая гибкость и масштабируемость системы. Использование внешних ключей для связи таблиц с core\_customuser как центральной позволяет поддерживать логическую структуру базы данных, необходимую для сложных операций, таких как управление дежурствами, чатами и аналитикой, без дублирования данных.

Для удобного использования данной системы был разработан интерфейс, который позволяет пользователям комфортно и эффективно использовать данной приложение.

Интерфейс приложения построен на основе единого шаблона base.html, который определяет общую структуру всех страниц: шапку с навигационным меню, основное содержимое и подвал. Навигационное меню, расположенное в верхней части страницы, включает такие элементы, как "График дежурств", "Мои профиль", "Аналитика", "Рабочий день", "Чат" и "Выйти".

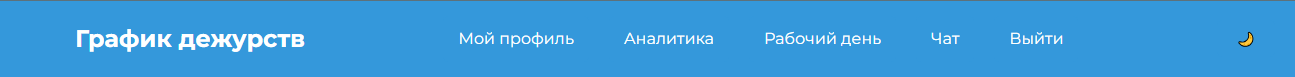


Рисунок 22 – Вид навигационного меню клиентской части приложения

Страница входа данного приложения простая и минималистичная, которая содержит поля для ввода имени пользователя и пароля, а также кнопкой "Войти". Форма расположена по центру страницы, что обеспечивает удобство и акцент на ключевом действии. Ссылка "Зарегистрироваться" направляет пользователя на страницу регистрации, улучшая навигацию.

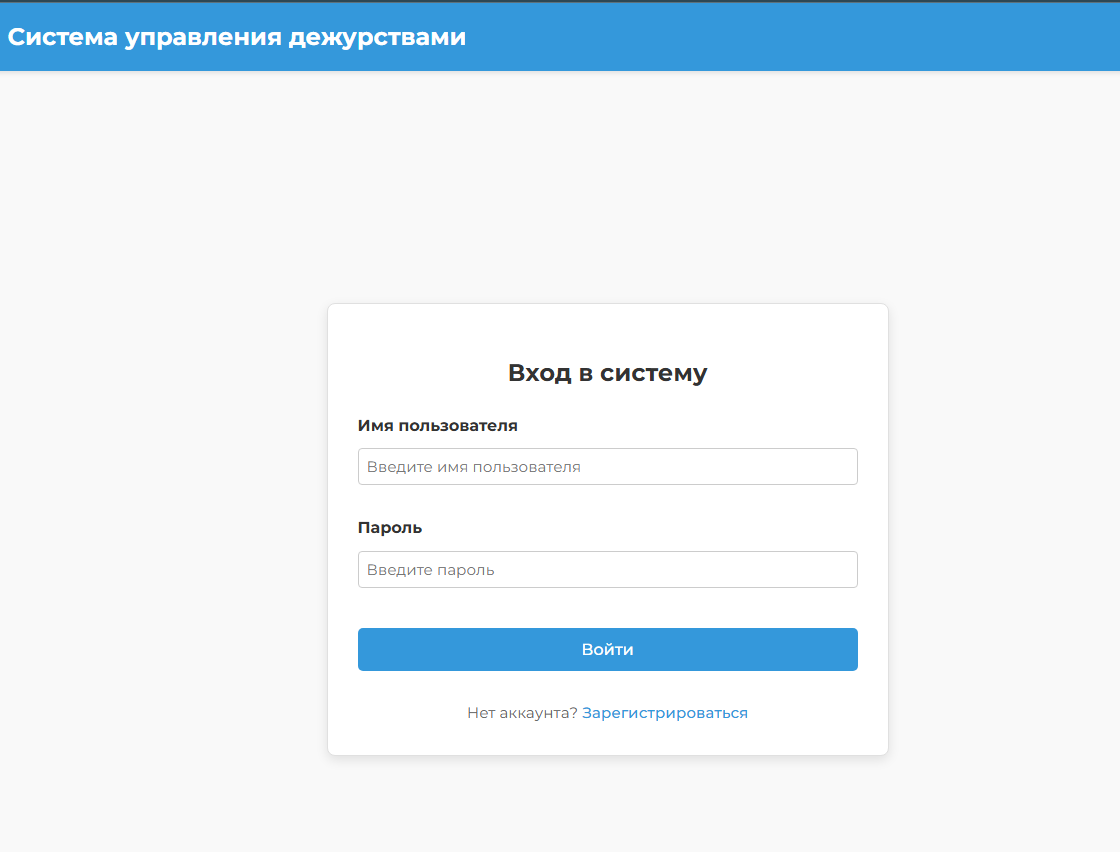


Рисунок 23 – Интерфейс входа в систему

Страница регистрации содержит форму с полями для имени пользователя, имени, фамилии, email и пароля (с подтверждением). Поля организованы вертикально с чёткими метками, что облегчает заполнение данных. Кнопка "Зарегистрироваться" выделена синим цветом, акцентируя внимание на завершении процесса.

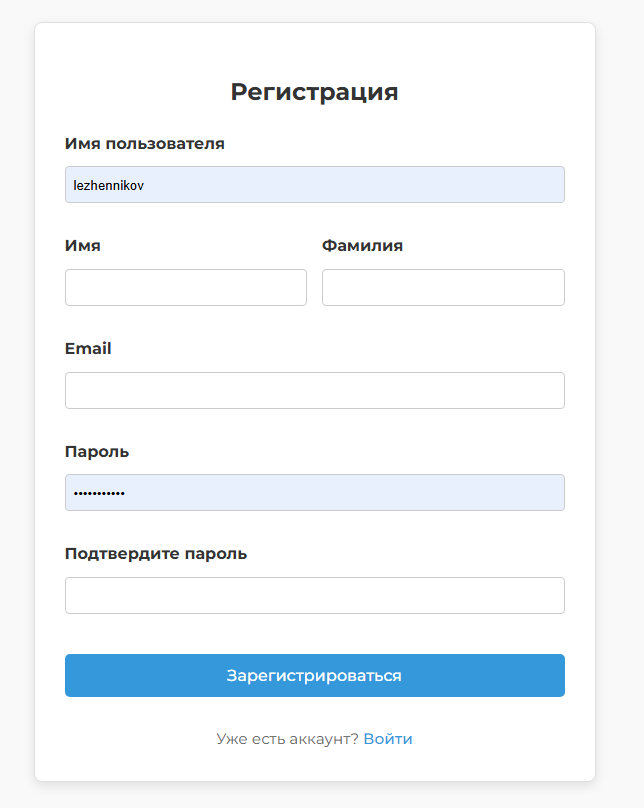


Рисунок 24 – Интерфейс регистрации

Раздел «Мой профиль» отображает информацию о пользователе и список дежурств за все время. Профиль представлен в виде карточки с чёткой структурой: метки слева, значения справа, что упрощает чтение.

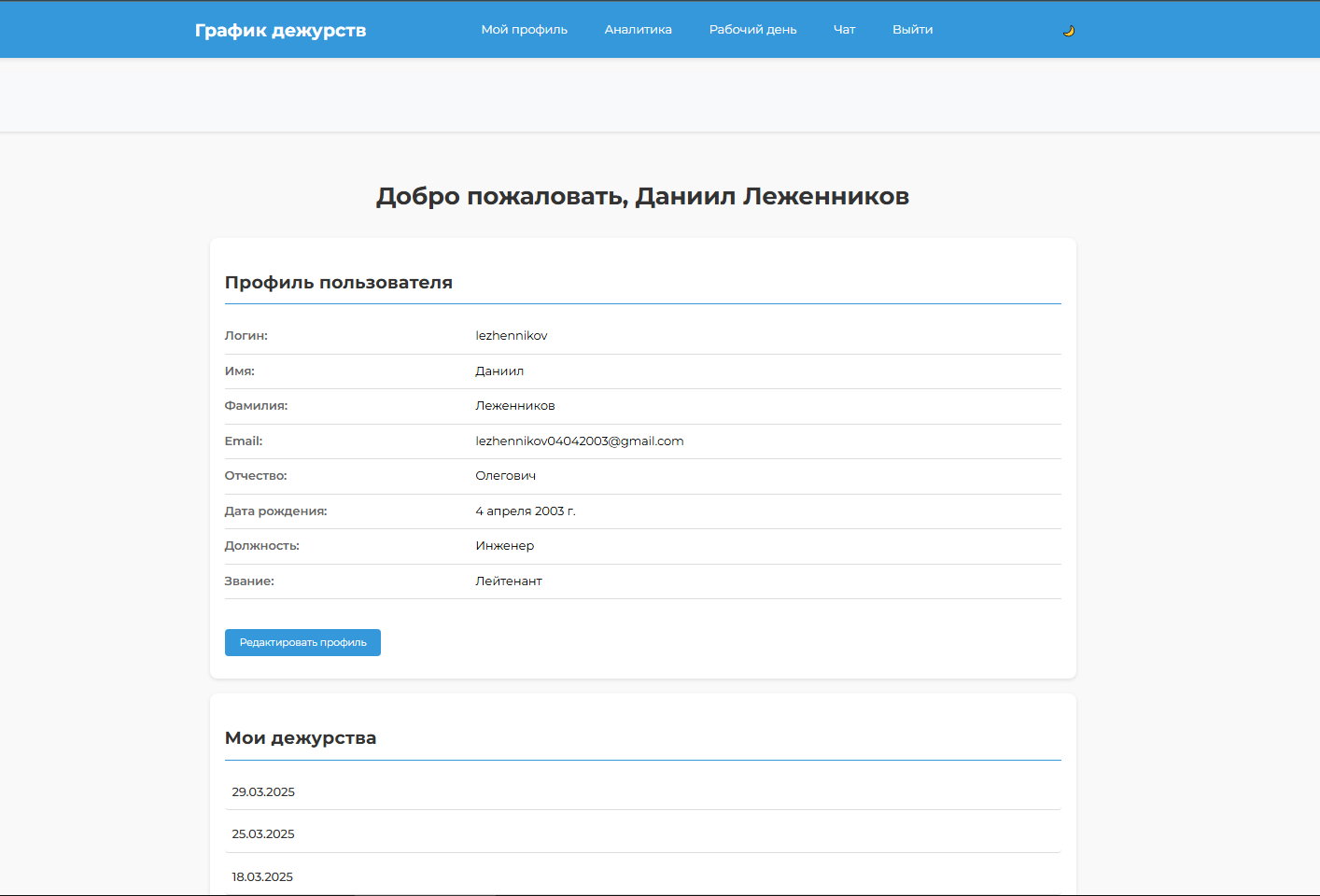


Рисунок 25 – Интерфейс раздела «Мой профиль»

Кнопка "Редактировать профиль" открывает модальное окно для изменения данных, что минимизирует необходимость перехода на отдельную страницу, повышая удобство. Модальное окно для редактирования профиля центрируется на экране, имеет интуитивно понятную форму с полями, соответствующими данным профиля, и кнопку "Сохранить изменения", обеспечивая удобный и быстрый процесс редактирования.

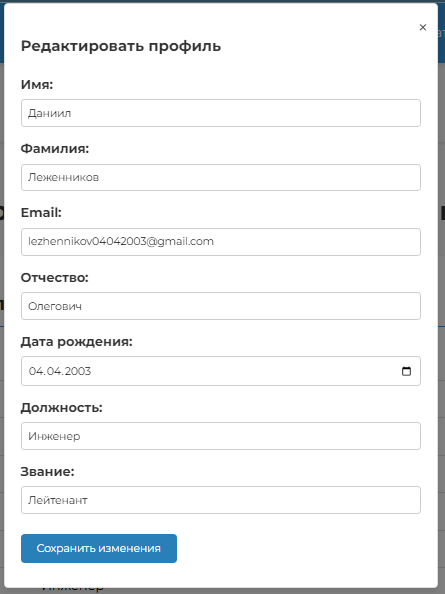


Рисунок 26 – Интерфейс окна для редактирования данных

Страница «Аналитика» предоставляет статистику дежурств пользователя (по месяцам, кварталам и годам) в виде карточек с числовыми значениями и часов. Календарь дежурств за текущий месяц отображается в виде таблицы с выделением дней дежурств зелёным цветом, что визуально отличает их от обычных дней.

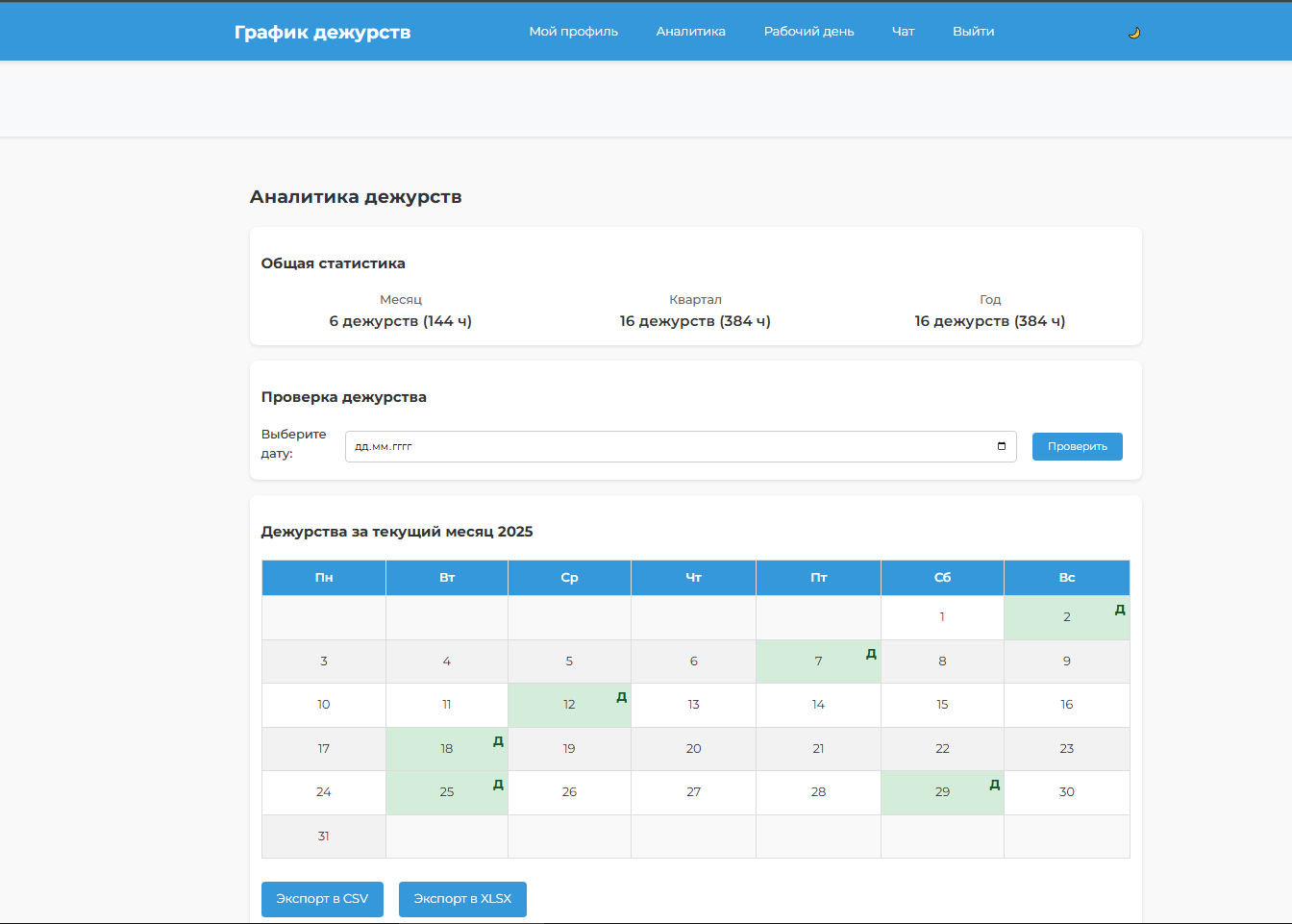


Рисунок 27 – Интерфейс страницы «Аналитика»

Форма для проверки дежурства на конкретную дату и кнопки экспорта данных в CSV/XLSX добавляют функциональности, делая интерфейс гибким и полезным для анализа. Дизайн страниц аналитики организован с акцентом на читаемость и удобство навигации.

Раздел рабочий день включает таймер рабочего дня (с кнопками "Старт" и "Сброс"), блок для отметки посещения и экспорта данных в табель учета рабочего времени, а также раздел для задач пользователя. Таймер отображается в отдельной карточке с крупным шрифтом, что делает его заметным и удобным для отслеживания времени.

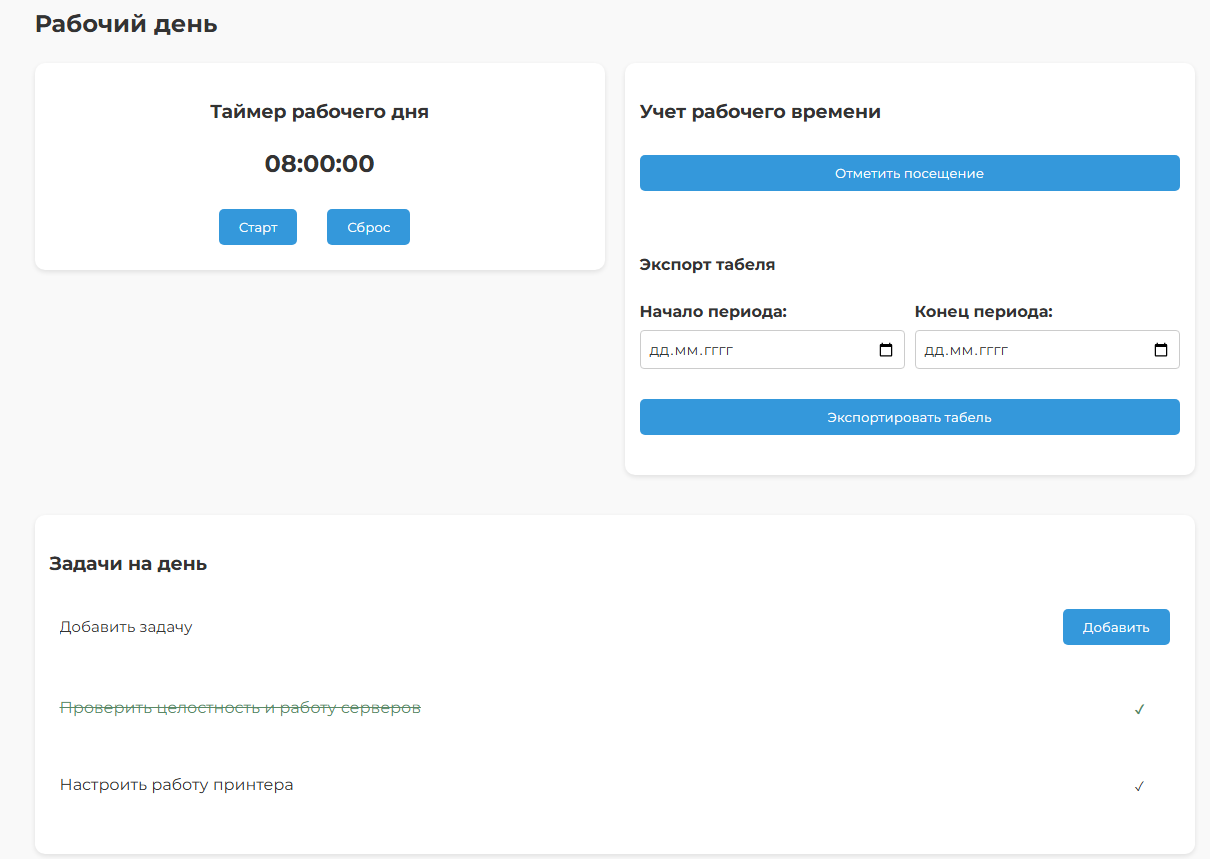


Рисунок 28 – Интерфейс раздела «Рабочий день»

Кнопки "Отметить посещение" и "Экспорт" расположены рядом, что упрощает доступ к ключевым функциям. Список задач представлен в виде списка с возможностью добавления и отметки выполнения, что интегрирует управление задачами в рабочий процесс.

Страница чата представляет собой простой интерфейс для общения пользователя с администратором. Сообщения отображаются в виде списка, разделённого на отправленные (справа) и полученные (слева), с указанием времени отправки, что обеспечивает чёткость и удобство чтения.

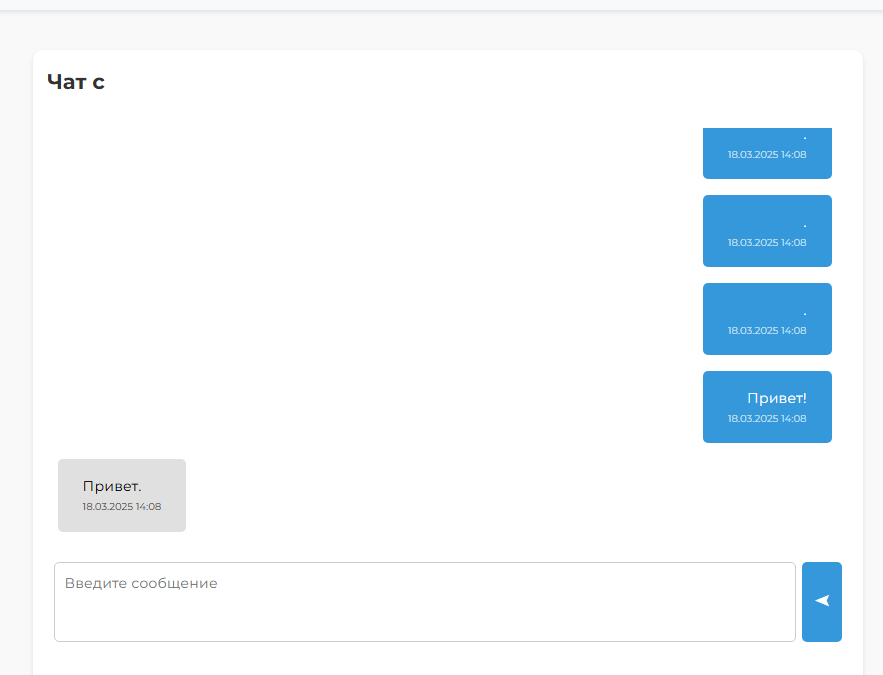


Рисунок 29 – Интерфейс страницы «Чат»

## **2.2. Реализация функций web-приложения**

Клиентская часть web-приложения для учета рабочего времени и графика дежурств разработана с использованием фреймворка Django, что позволяет эффективно создавать и поддерживать сложные веб-приложения. В структуре клиентской части выделяются несколько ключевых компонентов, каждый из которых выполняет свою функцию.

Модели данных отвечают за хранение и управление информацией, обеспечивая надежное взаимодействие с базой данных. Представления (views) обрабатывают запросы от пользователя, реализуя логику взаимодействия с сервером. Шаблоны интерфейса отвечают за формирование внешнего вида страниц, предоставляя пользователю удобный и понятный интерфейс. Статические файлы поддерживают визуальное оформление и функциональное взаимодействие с пользователем, улучшая восприятие и обеспечивая дополнительные возможности для работы с приложением.

Все основные данные о пользователях и их расписаниях хранится в файле models.py. В модели CustomUser содержится подробная информация о каждом пользователе, включая фамилию, имя, отчество, адрес электронной почты, должность, звание и дату рождения.

Листинг 1

class CustomUser(AbstractUser):  
middle\_name = models.CharField(max\_length=150, blank=True, verbose\_name="Отчество")  
birth\_date = models.DateField(null=True, blank=True, verbose\_name="Датарождения")  
 position = models.CharField(max\_length=100, blank=True, verbose\_name="Должность")  
 rank = models.CharField(max\_length=100, blank=True, verbose\_name="Звание")  
 REQUIRED\_FIELDS = ['email']

Эта модель помогает эффективно управлять данными о пользователях системы. Модель DutySchedule отвечает за хранение информации о дежурствах. Она позволяет связывать каждого пользователя с конкретными датами, на которые назначены дежурства.

Листинг 2

class DutySchedule(models.Model):  
 user = models.ForeignKey('CustomUser', on\_delete=models.CASCADE, related\_name='duties')  
 date = models.DateField()  
def \_\_str\_\_(self):  
return f"{self.user.username} - {self.date}"

Управление уведомлениями в системе осуществляется через модель Notification.Эта модель предназначена для отправки напоминаний пользователям о предстоящих дежурствах. Каждый пользователь, на которого назначено дежурство, получает уведомление заранее.

Листинг 3

class Notification(models.Model):  
 user = models.ForeignKey(CustomUser, related\_name='notifications', on\_delete=models.CASCADE)  
 message = models.TextField()  
 is\_read = models.BooleanField(default=False)  
created\_at = models.DateTimeField(auto\_now\_add=True)  
def \_\_str\_\_(self):  
return f"Уведомлениедля{self.user}: {self.message[:20]}"

Модель Notification сохраняет информацию о тексте уведомления и времени его создания. Это позволяет отслеживать момент формирования каждого уведомления. Такой подход обеспечивает точную отправку уведомлений пользователям в нужное время.

Обработка запросов реализована с помощью функций и классов, которые принимают входящие HTTP-запросы и возвращают нужные ответы. Например, функция home\_view отвечает за проверку предстоящих дежурств. Она анализирует дежурства на следующий день и, если уведомление еще не было создано, добавляет его в базу данных.

Листинг 4

fordutyinupcoming\_duties:  
message= f"Напоминание: у вас дежурство {duty.date.strftime('%d.%m.%Y')}."if not Notification.objects.filter(user=request.user, message=message).exists():  
Notification.objects.create(user=request.user, message=message)notifications = Notification.objects.filter(user=request.user, is\_read=False)

Этот код проверяет, назначены ли пользователю дежурства на следующий день. Если уведомление еще не было создано, оно генерируется. Уведомления предназначены только для обычных пользователей, в то время как администраторы не участвуют в этом процессе.

У пользователя есть возможность просматривать свои личные данные включая: ФИО, адрес электронной почты, дату рождения, должность и звание. Эти данные доступны для редактирования, для этого создана форма, которая позволяет обновить информацию. После сохранения изменений данные автоматически отображаются в профиле.Пользователь также может просмотреть список всех дежурств за весь период.

Листинг5

data = json.loads(request.body)  
logger.info(f"Received data: {data}") user = CustomUser.objects.get(id=user\_id)user.first\_name = data.get('first\_name', user.first\_name)  
user.last\_name = data.get('last\_name', user.last\_name)  
user.email = data.get('email', user.email)  
user.middle\_name = data.get('middle\_name', user.middle\_name)  
user.birth\_date = data.get('birth\_date') or None  
user.position = data.get('position', user.position)  
user.rank = data.get('rank', user.rank)if not user.first\_nameor not user.last\_nameor not user.email:  
return JsonResponse({'success': False, 'error': 'Заполнитеобязательныеполя.'}, status=400)  
user.save()

Пользователю предоставляется возможность анализировать свои дежурства. Аналитика включает данные о количестве дежурств, которые были назначены в текущем месяце, квартале и за год. Вся эта информация переводится в часы за эти же периоды. Это позволяет пользователю следить за динамикой своих дежурств и иметь полное представление о своей занятости в разные периоды времени.

Листинг 6

def analytics\_view(request):  
user = request.user  
current\_month= now().month  
current\_year= now().yearmonthly\_duties= DutySchedule.objects.filter(  
user=user,  
date\_\_year=current\_year,  
date\_\_month=current\_month  
).count()  
monthly\_hours= monthly\_duties\* 24quarterly\_duties= DutySchedule.objects.filter(  
user=user,  
date\_\_year=current\_year,  
date\_\_month\_\_gte=max(1, current\_month- 2),  
date\_\_month\_\_lte=current\_month  
).count()  
quarterly\_hours= quarterly\_duties\* 24yearly\_duties= DutySchedule.objects.filter(  
user=user,  
date\_\_year=current\_year  
).count()  
yearly\_hours= yearly\_duties\* 24

Пользователь может проверить, назначено ли у него дежурство на определенную дату. Для этого достаточно ввести нужную дату в запросе. Система немедленно выполнит проверку и сообщит, есть ли дежурство на этот день.

Листинг 7

specific\_date= request.GET.get('date')  
if specific\_date:  
duties\_on\_date= DutySchedule.objects.filter(user=user, date=specific\_date).exists()  
else:  
duties\_on\_date= None

Можно ознакомиться с перечнем дежурств за текущий месяц. Он упорядочен по датам, начиная с самых последних. Для удобства работы с информацией предусмотрены два варианта экспорта: в формате CSV и XLSX. Экспорт в CSV осуществляется через стандартную библиотеку csv, а для XLSX используется библиотека openpyxl.

Листинг 8

if 'export\_csv' in request.GET:  
response = HttpResponse(content\_type='text/csv')  
response['Content-Disposition'] = 'attachment; filename="duties.csv"'  
writer = csv.writer(response)  
writer.writerow(['Дата'])  
for duty in all\_duties:  
writer.writerow([duty.date])  
return response

Листинг 9

if 'export\_xlsx' in request.GET:  
wb= Workbook()  
ws= wb.active  
ws.append(['Дата'])  
for duty in all\_duties:  
ws.append([duty.date])  
buffer = BytesIO()  
wb.save(buffer)  
buffer.seek(0)  
response = HttpResponse(buffer.read(), content\_type='application/vnd.openxmlformats-officedocument.spreadsheetml.sheet')  
response['Content-Disposition'] = 'attachment; filename="duties.xlsx"'  
return response

С помощью раздела «Рабочий день» пользователь сможет эффективно отслеживать продолжительность рабочего дня и отслеживать потраченное время. Также предусмотрена возможность создания и управления списком задач на день. Задачи добавляются через текстовое поле: при вводе текста задачи и отправке формы создается новая запись, которая автоматически привязывается к текущему пользователю и дате. Для каждой задачи можно изменить статус выполнения – отметить, как выполненную или вернуть в работу.

Листинг 10

def workday\_view(request):  
if request.method == 'POST':  
if 'add\_task' in request.POST:  
task\_text= request.POST.get('task\_text')  
if task\_text:  
Task.objects.create(user=request.user, text=task\_text)  
elif'toggle\_task' in request.POST:  
task\_id= request.POST.get('task\_id')  
task = Task.objects.get(id=task\_id, user=request.user)  
task.completed = not task.completed  
task.save()  
return JsonResponse({'success': True})  
tasks = Task.objects.filter(user=request.user, date=now().date())  
context = {  
'tasks': tasks,  
 }  
return render(request, 'workday.html', context)

Для обеспечения комфортного и эффективного взаимодействия между пользователями и администратором в системе предусмотрен функционал чата. Он предназначен для обмена текстовыми сообщениями, позволяя пользователям и администратору вести диалоги в реальном времени. Все сообщения сохраняются в базе данных с указанием отправителя, получателя и временной метки. История переписки загружается при открытии чата и отображается в хронологическом порядке. Непрочитанные сообщения автоматически помечаются как прочитанные при загрузке страницы чата.

Листинг 11

def chat\_view(request, receiver\_id):  
receiver = get\_object\_or\_404(CustomUser, id=receiver\_id)  
messages\_list = ChatMessage.objects.filter(  
 models.Q(sender=request.user, receiver=receiver) |  
 models.Q(sender=receiver, receiver=request.user)  
 ).order\_by('timestamp')ChatMessage.objects.filter(sender=receiver, receiver=request.user, is\_read=False).update(is\_read=True)  
if request.method == 'POST':  
message\_text = request.POST.get('message')  
if message\_text:  
 ChatMessage.objects.create(sender=request.user, receiver=receiver, message=message\_text)  
return redirect('chat', receiver\_id=receiver\_id)  
context = {'receiver': receiver,'messages': messages\_list,}  
return render(request, 'chat.html', context)

Листинг 12

def chat\_with\_admin(request):  
if request.user.is\_superuser:  
return redirect('chat\_list') admin = CustomUser.objects.filter(is\_superuser=True).first()  
if not admin:  
 messages.error(request, "Администраторненайден.")  
return redirect('user\_dashboard')  
return redirect('chat', receiver\_id=admin.id)

Пользователь может ввести текст сообщения в соответствующую форму и отправить его. Каждое новое сообщение сохраняется в базе данных и мгновенно отображается в интерфейсе. Все сообщения сопровождаются информацией о времени отправки и отправителе.

## **2.3 Тестирование**

Теперь после реализации всех функций, необходимо их протестировать. Для этого были созданы тесты на языке Python, которые используют классы, чтобы выполнить качественное тестирование приложения.

Для начала проводятся тесты для моделей, которые представляют структуру данных приложения. Тестирование выполняется с помощью класса TestCase, который дает изолированную среду для тестов с отдельной базой. Подход заключается в создании примеров моделей, настройке их свойств и правильности сохранения данных. Для проверки результатов использованы встроенные методы assert, такие как assertEqual, assertTrue и assertIsInstance, которые позволяют убедиться в правильности работы моделей.

Листинг 13

class ModelTests(TestCase):

def setUp(self):

self.user = CustomUser.objects.create\_user(

username='testuser',

password='testpassword',

email='test@example.com')

def test\_custom\_user\_creation(self):

self.assertEqual(self.user.username, 'testuser')

self.assertEqual(self.user.email, 'test@example.com')

self.assertTrue(self.user.check\_password('testpassword'))

def test\_duty\_creation(self):

duty = Duty.objects.create(

user=self.user,

date=timezone.now().date(),

status='Активно')

self.assertEqual(duty.user, self.user)

self.assertEqual(duty.status, 'Активно')

self.assertIsInstance(duty.date, date)

Данный класс проверяет корректность создания и работу моделей. Тесты проверяют правильность сохранения данных и связи между моделями.

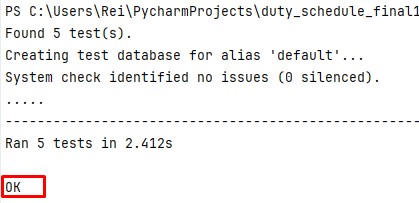


Рисунок 30 – Результаты тестирования моделей

Теперь тестируются представления Django, которые обрабатывают HTTP-запросы и возвращают ответы. Основным инструментом будет Django Test Client, который эмулирует HTTP-запросы к представлениям. В данном тесте имитируются запросы GET и POST, осуществляется проверка кодов ответа HTTP, эмулируются перенаправления и содержимое ответов. Для тестирования защищенных представлений используется функция login, которая аутентифицирует тестовый клиент. Тесты проверяют как успешные исходы, так и случаи с ошибками.

Листинг 14

class ViewTests(TestCase):

def setUp(self):

self.user = CustomUser.objects.create\_user(

username='testuser',

password='testpassword',

email='test@example.com'

)

self.client = Client()

def test\_login\_view\_get(self):

"""Тест GET-запроса к представлению входа."""

response = self.client.get(reverse('login'))

self.assertEqual(response.status\_code, 200)

self.assertTemplateUsed(response, 'login.html')

def test\_login\_view\_post\_valid(self):

response = self.client.post(reverse('login'), {

'username': 'testuser',

'password': 'testpassword'

})

self.assertRedirects(response, reverse('home'))

self.assertTrue(response.wsgi\_request.user.is\_authenticated)

Данный класс проверяет корректность работы представлений для аутентификации пользователей, включая вход в систему и регистрацию.

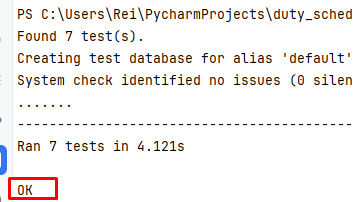


Рисунок 31 – Результаты тестирования представлений

Данное тестирование подтверждает, что все представления данного приложения корректно обрабатывают пользовательские запросы. Это означает, что аутентификация работает правильно, пользователи могут успешно регистрироваться и входить в систему, а защищенные ресурсы доступны только авторизованным пользователям.

Тестирование форм, которые обеспечивают правильность ввода и получения данных перед сохранением в базу, заключается в создании форм с различными данными и проверке их валидности и ошибок. Для этого используются методы is\_valid для проверки валидации и доступ к словарю с ошибками. Тесты проверяют разные случаи: правильные данные, пустые поля и некорректные форматы данных.

Листинг 15

class FormTests(TestCase):

def setUp(self):

self.user = CustomUser.objects.create\_user(

username='testuser',

password='testpassword',

email='test@example.com'

)

def test\_login\_form\_valid(self):

"""Тест валидной формы входа."""

form = LoginForm(data={

'username': 'testuser',

'password': 'testpassword'

})

self.assertTrue(form.is\_valid())

def test\_login\_form\_empty(self):

form = LoginForm(data={})

self.assertFalse(form.is\_valid())

self.assertEqual(len(form.errors), 2)

self.assertIn('username', form.errors)

self.assertIn('password', form.errors)

Этот тест проверяет корректность работы валидации во всех формах приложения.

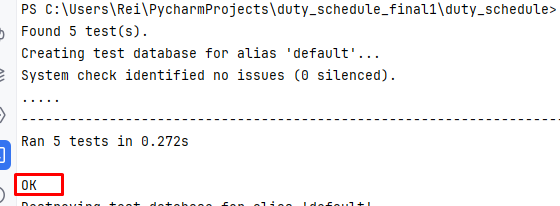


Рисунок 32 – Результаты тестирования форм

Тестирования подтверждает корректную работу валидации во всех формах приложения. Приложение защищено от некорректных данных, которые могут быть введены пользователем.

Тестирование взаимодействия между различными компонентами приложения, позволяет выявить проблемы на стыках между ними. Основным инструментом является Django Test Client, но тесты охватывают более сложные исходы, которые включают несколько запросов и взаимодействие между различными моделями и представлениями. Тестовый клиент, используется для имитации последовательности действий пользователя.

Листинг 16

class IntegrationTests(TestCase):

def setUp(self):

self.client = Client()

self.user\_data = {

'username': 'newuser',

'email': 'newuser@example.com',

'password1': 'complex\_password',

'password2': 'complex\_password'

}

def test\_register\_and\_login(self):

self.client.post(reverse('register'), self.user\_data)

self.assertTrue(CustomUser.objects.filter(username='newuser').exists())

self.client.logout()

response = self.client.post(reverse('login'), {

'username': 'newuser',

'password': 'complex\_password'

})

self.assertRedirects(response, reverse('home'))

self.assertTrue(response.wsgi\_request.user.is\_authenticated)

Данный класс проверяет полные процессы работы с приложением, такие как регистрация пользователя, аутентификация, создание о отметки о задачах, Просмотр дежурств, экспорт дежурств и табеля, получение и отправка сообщений.

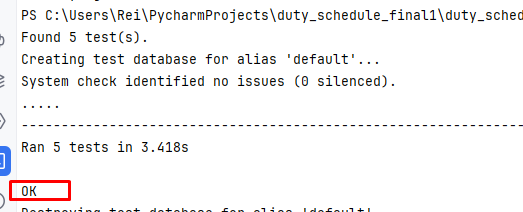


Рисунок 33 – Результаты теста взаимодействия между компонентами приложения

Данный тест подтверждает, что все компоненты приложения корректно взаимодействуют друг с другом. Это позволяет понять, что пользователи могут выполнять полный набор процессов. Успешное завершение всех тестов, подтверждает корректную работу приложения. Ключевые возможности функционируют правильно и без ошибок. Приложение подготовлена к работе в реальных условиях.

## **ВЫВОДЫ ПО ГЛАВЕ 2**

В проектном разделе разработано веб-приложение "График дежурств", предназначенное для учета рабочего времени, управления дежурствами и обеспечения эффективного взаимодействия между пользователями и администратором. Приложение предоставляет пользователю удобный интерфейс для работы с персональными данными, аналитикой дежурств, задачами, уведомлениями и чатом, а также функционал экспорта данных в форматы CSV и XLSX для интеграции с внешними системами, такими как Google Calendar.

Для реализации веб-приложения были использованы следующие программные средства: фреймворк Django, обеспечивающий гибкость и удобство при разработке сложных систем, а также статические файлы CSS и JavaScript для визуального оформления и интерактивности. Django позволил эффективно организовать структуру приложения, разделив его на функциональные блоки: модели для работы с данными, представления для обработки запросов и шаблоны для формирования интерфейса.

Для достижения целей исследования было разработано веб-приложение, обеспечивающее контроль дежурств пользователем, учет рабочего времени (с использованием таймера и отметки посещения), постановку задач для пользователя (добавление, редактирование и отметка выполнения задач), экспорт дежурств в форматы CSV и XLSX, а также анализ активности через статистику дежурств и уведомления о предстоящих событиях. Приложение разработано с учетом безопасности (защита от CSRF-атак, использование HTTPS) и удобства использования, включая поддержку темной темы и сохранение данных в localStorage для сохранения состояния при перезагрузке страниц.

В последнем пункте главы функциональные возможности приложения были успешно протестированы. В результате тестирования было выявлено, что программный продукт работает корректно, все ключевые функции (вход, регистрация, аналитика, управление задачами, чат) соответствуют заявленным требованиям, а интерфейс обеспечивает удобство и интуитивность взаимодействия для пользователей. Тестирование подтвердило готовность приложения к использованию в реальных условиях.

# **ЗАКЛЮЧЕНИЕ**

Современные системы учета рабочего времени и управления дежурствами играют важную роль в повышении эффективности работы организаций, обеспечивая точный контроль трудозатрат, оптимизацию процессов и улучшение взаимодействия между сотрудниками. Разработка таких систем с использованием удобных и функциональных интерфейсов становится перспективным направлением для автоматизации управления персоналом.

В рамках выпускной квалификационной работы было разработано веб-приложение "График дежурств", позволяющее пользователям управлять своим рабочим временем, дежурствами, задачами и взаимодействовать с администратором через чат. В ходе выполнения работы был проведен анализ предметной области, изучены существующие решения для учета рабочего времени, а также подобраны программные средства для реализации проектного решения.

Задачи, поставленные в начале работы, выполнены в полном объеме:

* проанализированы требования к функционалу клиентской части системы учета рабочего времени и графика дежурств;
* разработан интерфейс для просмотра графика дежурств, аналитики и экспорта данных;
* реализован механизм уведомлений, информирующих пользователей о предстоящих дежурствах;
* обеспечена возможность редактирования личных данных пользователей через личный кабинет;
* реализован раздел с таймером для отслеживания рабочего времени, отметкой посещения и управлением задачами;
* обеспечена функциональность чата между пользователем и администратором для оперативного взаимодействия и решения вопросов;
* протестирован разработанный модуль на соответствие требованиям удобства, производительности и корректности работы.

Таким образом, в результате решения поставленных задач была достигнута цель выпускной квалификационной работы – разработан пользовательский модуль системы учета рабочего времени и графика дежурств, обеспечивающий удобное взаимодействие сотрудников с системой для просмотра, анализа и управления их расписанием.

# **СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Django documentation: сайт. – URL: https://docs.djangoproject.com/en/5.2/ (дата обращения: 02.02.2025). – Текст: Электронный
2. FastAPI Documetation: сайт. – URL: https://fastapi.tiangolo.com/ (дата обращения: 03.04.2025). – Текст: Электронный
3. Flask Documetation: сайт. – URL: https://flask.palletsprojects.com/en/latest/ (дата обращения: 09.04.2025). – Текст: Электронный
4. Java Platform: сайт. – URL: https://docs.oracle.com/en/java/ (дата обращения: 15.03.2025). – Текст: Электронный
5. JavaScript: Документация сайт. – URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата обращения: 11.03.2025). – Текст: Электронный
6. Laravel: сайт. – URL: https://laravel.com/docs/12.x (дата обращения: 11.03.2025). – Текст: Электронный
7. Matplotlib: сайт. – URL: https://matplotlib.org/stable/users/index.html (дата обращения: 09.03.2025). – Текст: Электронный
8. Metanit: сайт: [Руководство по SQLite]. – URL: https://metanit.com/sql/sqlite/ (дата обращения: 09.03.2025). – Текст: Электронный
9. MongoDB Documentation: сайт. – URL: https://www.mongodb.com/docs/ (дата обращения: 09.03.2025). – Текст: Электронный
10. MySQL Documentation: сайт. – URL: https://dev.mysql.com/doc/ (дата обращения: 10.03.2025). – Текст: Электронный
11. NumPy documentation: сайт. – URL: https://numpy.org/doc/stable/ (дата обращения: 15.03.2025). – Текст: Электронный
12. Pandas documentation: сайт. – URL: https://pandas.pydata.org/docs/ (дата обращения: 01.04.2025). – Текст: Электронный
13. PHP Manual: сайт. – URL: https://www.php.net/manual/en/index.php#index (дата обращения: 02.04.2025). – Текст: Электронный
14. PostgreSQL Documentation: сайт. – URL: https://www.postgresql.org/docs/ (дата обращения: 10.03.2025). – Текст: Электронный
15. Python 3.13.3 documentation: сайт. – URL: https://docs.python.org/3/ (дата обращения: 09.03.2025). – Текст: Электронный
16. Spring Framework Documentation: сайт. – URL: https://docs.spring.io/spring-framework/reference/ (дата обращения: 12.02.2025). – Текст: Электронный
17. SQLite Documentation: сайт. – URL: https://sqlite.org/docs.html (дата обращения: 09.03.2025). – Текст: Электронный
18. SQLite: как организовывать таблицы // medium : сайт. – URL: https://goo.su/UQC5OiS (дата обращения: 10.03.2025)
19. Symfony Documentation: сайт. – URL: https://symfony.com/doc/current/index.html (дата обращения: 09.03.2025). – Текст: Электронный
20. Tornado Documentation: сайт. – URL: https://www.tornadoweb.org/en/stable/ (дата обращения: 09.03.2025). – Текст: Электронный
21. tutorialspoint.com: сайт: [SQLite Tutorial]. – URL: https://www.tutorialspoint.com/sqlite/index.htm (дата обращения: 14.04.2025). – Текст: Электронный
22. TypeScript Documentation: сайт. – URL: https://www.typescriptlang.org/docs/ (дата обращения: 09.03.2025). – Текст: Электронный
23. Веб-фреймворк Django (Python) // developer.mozilla.org: сайт. – URL: https://developer.mozilla.org/ru/docs/Learn\_web\_development/Extensions/Server-side/Django (дата обращения: 22.03.2025)
24. Вепрев, С. Б. Введение в Django : краткий учебный курс / С. Б. Вепрев, В. В. Крылов. - Москва : ИНТУИТ, 2016. - 261 с. - Текст : электронный. - URL: https://znanium.ru/catalog/product/2138301 (дата обращения: 10.04.2025). – Режим доступа: по подписке.
25. Гаско, Р. Простой Python просто с нуля / Р. Гаско ; под ред. Н. Ю. Комлева. - Москва : СОЛОН-ПРЕСС, 2023. - 256 с. - (Серия «Программирование»). - ISBN 978-5-91359-334-4. - Текст : электронный. - URL: https://znanium.ru/catalog/product/2185854 (дата обращения: 11.03.2025). – Режим доступа: по подписке.
26. Зайцева, О. С. Технологии разработки web-ресурсов : учебное пособие : [16+] / О. С. Зайцева ; Тюменский индустриальный университет. – Тюмень : Тюменский индустриальный университет, 2020. – 75 с. : ил., табл. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=611103 (дата обращения: 10.02.2025). – ISBN 978-5-9961-2274-5. – Текст : электронный.
27. Марухленко, А. Л. Разработка защищённых интерфейсов Web-приложений : учебное пособие : [16+] / А. Л. Марухленко, Л. О. Марухленко, М. А. Ефремов. – Москва ; Берлин : Директ-Медиа, 2021. – 175 с. : ил., табл. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=599050 (дата обращения: 12.02.2025). – Библиогр. в кн. – ISBN 978-5-4499-1676-1. – DOI 10.23681/599050. – Текст : электронный.
28. Меле, А. Django 4 в примерах / А. Меле. - Москва : ДМК Пресс, 2023. - 802 с. - ISBN 978-5-93700-204-4. - Текст : электронный. - URL: https://znanium.ru/catalog/product/2150534 (дата обращения: 12.03.2025). – Режим доступа: по подписке.
29. Побиянская А.В., Кипервар Е.А., Дубровин А.М. Системы контроля и учета рабочего времени как направление повышения производительности труда // Экономика труда. - 2021. - №6. - С. 631-642.
30. Рамальо, Л. Python - к вершинам мастерства: Лаконичное и эффективное программирование: практическое руководство / Л. Рамальо ; пер. с англ. А. А. Слинкина. - 2-е изд. - Москва: ДМК Пресс, 2022. - 900 с. - ISBN 978-5-97060-885-2. - Текст: электронный. - URL: https://znanium.ru/catalog/product/2109578 (дата обращения: 16.03.2025). – Режим доступа: по подписке.
31. С.В. Бегичева Автоматизированные системы учета рабочего времени сотрудников // Экономика и бизнес: теория и практика. - 2023. - С. 12-16.
32. Шелудько, В. М. Основы программирования на языке высокого уровня Python : учебное пособие : [16+] / В. М. Шелудько. – Ростов-на-Дону ; Таганрог : Южный федеральный университет, 2017. – 147 с. : ил. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=500056 (дата обращения: 01.04.2025). – Библиогр. в кн. – ISBN 978-5-9275-2649-9. – Текст : электронный.
33. Шкодина, Т. А. Алгоритмы и структуры данных в Python : лабораторный практикум : учебное пособие : [16+] / Т. А. Шкодина ; Ростовский государственный экономический университет (РИНХ), Кафедра информационных систем и прикладной информатики. – Ростов-на-Дону : Издательско-полиграфический комплекс РГЭУ (РИНХ), 2022. – 80 с. : ил., табл. – Режим доступа: по подписке. – URL: https://biblioclub.ru/index.php?page=book&id=704757 (дата обращения: 10.04.2025). – ISBN 978-5-7972-3037-3. – Текст : электронный.